Mahy Montesdeoca Hernández

Cibersecurity & hacking

[Informe de Lenguajes, Paradigmas y Estándares de Programación]

- **Introducción**

La existencia de los lenguajes, paradigmas y estándares en el mundo de la programación es muy importante, ya que estos nos dan las facilidades para manejar, controlar y mejorar las tecnologías de nuestro entorno. Nos facilitan la comunicación con las máquinas por así decirlo y son las responsables del buen funcionamiento de estas.

-**Tipos de lenguaje de programación**

Lenguajes de programación de nivel bajo: Este tipo de lenguaje hace referencia a aquel que actúa directamente en el hardware y se utiliza para programar tareas o funciones muy importantes del sistema operativo. Este no se refiere a un tipo de lenguaje en específico, sino que engloba algunos lenguajes diferentes como pueden ser binario, lenguaje en máquina y lenguajes ensambladores.

En general estos tipos de lenguaje se usan para organizar y dar órdenes para que actúen de manera eficiente tareas para el hardware o para programar sistemas operativos. Son muy efectivos y precisos.

Lenguajes de programación de nivel medio: Son aquellos caracterizados por combinar el lenguaje de nivel bajo con el de nivel alto, es decir, que esto permite a la persona encargada de la programación tener la efectividad de los lenguajes de bajo nivel, sin estar la complejidad de este y tiene mejor control que el lenguaje de nivel alto. Algunos ejemplos de lenguajes de nivel medio son C++, java y Python.

Estos se usan para distintas aplicaciones como pueden ser hojas de cálculos, sistemas gestores de bases de datos o para crear sistemas operativos ya que permiten un manejo abstracto debido a la mezcla de lenguaje de nivel bajo y nivel alto.

Lenguaje de programación de nivel alto: Es aquel que se parece mucho más que el resto al lenguaje humano y no tanto al binario, su función principal es que se puede ejecutar el mismo programa en distintas máquinas, es decir, que sea independiente de un hardware específico. Algunos ejemplos de lenguajes de nivel alto son Ruby, JavaScript y switch.

- **Paradigmas de Programación**

Existen diversos paradigmas de programación, a continuación, voy a nombrar los principales y a explicarlos.

Programación imperativa- Este es el paradigma de programación más antiguo, y consta de seguir cadenas de acciones una tras otra que se le ordenan para llegar al resultado deseado. Para organizar estas cadenas, se hace uso de bucles o estructuras anidadas en el código, cabe mencionar que este sistema es muy concreto y en general comparado con otros hay que escribir mucho más debido a que son más detallados.

Un ejemplo puede ser el lenguaje C, el cual es de bajo nivel como mencioné anteriormente y funciona ejecutando una secuencia en orden para llegar al resultado, otro ejemplo es Java, el cual sigue un paradigma imperativo.

Programación declarativa- Se caracteriza por su carácter instructivo y se especifica qué se quiere lograr, no como en el imperativo, donde se describe cómo lograrlo. En otras palabras, la persona encargada de programar describe el problema que sucede y deja en manos del sistema que este dé con la mejor forma de solucionar el problema. Es comúnmente utilizado en bases de datos y estos a menudo realizan optimizaciones automáticamente, lo que facilita más el trabajo.

Algunos ejemplos de lenguajes que lo usan son SQL, en donde se extrae datos de las bases de datos y HTML/css en donde se realiza la estructura de la página web y cómo se ve.

Programación orientada a objetos- Se distingue por el uso de clases (las cuales definen de manera genérica cómo van a ser los objetos de un determinado tipo) y objetos (son simplemente las entidades que participan en él), para estructurar el programa de software en pequeñas partes más simples y entendibles. La base de este paradigma es pensar más en los objetos que en la lógica pura, inspirándose más en el mundo que nos rodea que en el digital, esto hace mucho más viable la organización y gestión, por ejemplo, si estamos hablando de una tienda el sistema se centra mucho más en las entidades como los productos, fecha de caducidad... que en estructuras de datos.

Un ejemplo de lenguaje de programación que usa este paradigma es Python.

Programación funcional- Este tipo de programación es más diferente al resto ya que hace uso de funciones matemáticas como lambda y evita las confusiones/ cambios de datos. Sus características se basan en el uso de funciones únicamente puras, solo se puede generar valores y finalmente no hay bucles, a diferencia de la programación imperativa. Hay diversos ejemplos de lenguajes que usan este paradigma, como pueden ser haskell, que se basa en el cálculo lambda y Scala, que curiosamente combina la programación funcional con la orientada a objetos (se ejecuta en la máquina virtual de Java).

Programación lógica- En esta clase de programación, se tiene de base la lógica y es debido a esto que se tienen que declarar las cosas que quieres que se realicen y el ordenador decidirá la mejor forma de hacerlo. Se toma muy en cuenta el orden coherente de las sentencias o secuencias para que estas se cumplan de una manera más eficaz y rápida. Una característica de este modelo es que el programador que la usa tiene que fijarse si los enunciados puestos en el sistema ya que son la clave de que el resultado salga perfecto. Un ejemplo de lenguaje que usa el modelo lógico es Prolog , que se basa en hechos y reglas.

-Estándares de Programación

Los estándares de programación son aquellas reglas que se siguen para un lenguaje de programación. Esto hace que cuando varias personas trabajan en un mismo proyecto, todas tengan una idea clara sobre lo que sus compañeros están escribiendo y ejecutando en su parte del trabajo, lo que hace que esto sea algo muy esencial a la hora de trabajar, ya que, aunque el proyecto lo hagas tú solo, puedes necesitar ayuda, supervisión u opiniones de otro y para conseguir eso es necesario seguir una serie de estándares. Varias consecuencias de no seguir estas reglas pueden ser generar un ambiente de trabajo hostil, sin comunicación ni entendimiento por parte de los miembros y debates sobre qué proceso/ código está mejor expresado. La forma correcta de implementar estándares de programación es planeándolo e implementando estas normas, para que así se entienda todo a la perfección y se cree una buena armonía de trabajo. Es por esto que es tan beneficioso acordar unos estándares con los que todos estén cómodos y los usen de la forma correcta.

Existen algunos métodos de estándares conocidos, en Python, por ejemplo, se usa Python PEP8, guía en la cual se indican los modos estilísticos que se deben seguir en este lenguaje, algunas características de este son: hacer buen uso de espacios, los operadores como el “+” tienen que usar espacios, después de declarar una función se deben dejar dos espacios en blanco y usar líneas en blanco para agrupar pasos similares. Otro método conocido es el “JavaScript Standard Style”, cuyas

características son: Utilizar dos espacios para la sangría, utilizar comillas simples para cadenas excepto para evitar escapes, agregar un espacio después de las palabras claves y las comas deben llevar un espacio después de ser usadas.

-**Conclusión**

Haciendo este trabajo te quedan claras varias cosas, sobre todo la importancia de saber el tipo de lenguaje que se está usando, ya que existen diversos modelos para determinadas cosas que se quieran lograr. Hay que estar muy informado a la hora de usar estos lenguajes o paradigmas anteriormente mencionados, al igual que en el terreno de la comunicación, ya que hay que tener en cuenta que tipo de estándares de programación tiene o tendrá el equipo en el que te encuentras, esta última parte es muy importante ya que es la responsable de la calidad de trabajo y entendimiento que tendrás en tu área de trabajo, además el uso de estándares conocidos es un lujo, ya que así te podrás entender con muchas personas en el ámbito de la programación y no solo con tu equipo.

-Bibliografía

https://www.epitech-it.es/lenguaje-bajo-nivel/

https://miformacion.eu/blog/lenguaje-bajo-nivel-en-programacion/

https://abrirarchivos.info/tema/lenguaje-de-nivel-medio-que-es-y-como-funciona/

https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/programacion-imperativa/

https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/

https://openwebinars.net/blog/que-es-la-programacion-funcional-y-sus-caracteristicas/

https://keepcoding.io/blog/que-es-la-programacion-logica/

https://blog.thedojo.mx/2021/10/05/estandares-de-calidad-en-el-software.html

https://es.scribd.com/document/377254736/Estandares-de-Programacion

https://ellibrodepython.com/python-pep8

[Informe testing]

-Introducción

El testing y las pruebas de código son componentes muy importantes para el desarrollo del software, asegurando la calidad, confiabilidad, y funcionalidad del producto final. Estas pruebas son cruciales para identificar y corregir errores, de esta manera, el usuario tiene una mejor experiencia.

-Conceptos básicos, relevancia del testing y pruebas de código en el ciclo de vida del desarrollo del software.

Testing: Es el proceso de evaluación de un sistema para identificar errores o posibles defectos. Incluye varias actividades como pruebas de código, pruebas de unidad, y pruebas de integración entre otras.

Pruebas de código: Se centran específicamente en evaluar la calidad del código fuente, para que su funcionamiento sea correcto.

-Objetivos de realizar pruebas.

El objetivo principal es identificar y detectar defectos e irregularidades en el software, estas aseguran que el software cumple con los estándares de calidad establecidos, esto comprende la calidad, el rendimiento y la funcionalidad.

Las pruebas permiten validar que este software cumple con todos los requisitos mínimos para que se ejecute de forma correcta. La fiabilidad del software aumenta, reduciendo la probabilidad de que haya fallos.

En cuanto a los beneficios de realizar pruebas, el hecho de identificar de manera temprana los defectos, hace que el coste se reduzca al no alargar el momento en el que se encuentra el fallo.

Esto también mejora la productividad al ejecutarse de manera rápida y eficiente, permitiendo realizar pruebas frecuentes sin aumentar la carga de trabajo

-Tipos de pruebas.

Pruebas unitarias: Estas pruebas se centran en verificar que cada unidad o componente individual del software funcione como se espera. A menudo son escritas y ejecutadas por los propios desarrolladores. Su objetivo es garantizar que las partes más pequeñas del código, como las funciones o métodos operen correctamente de manera aislada. Sus herramientas populares son: PHPUnit, JUnit.

Pruebas de integración: Evalúan la interacción y cooperación entre diferentes unidades o módulos del software. Se aseguran de que los componentes integrados trabajen juntos sin problemas. Su objetivo es identificar los posibles problemas de interoperabilidad y garantizar el correcto funcionamiento de las interfaces entre módulos. Sus herramientas populares son: TstNG y Mockito.

Pruebas de sistema: Las pruebas de sistema evalúan el software al completo. Se hacen después de las pruebas de integración y estas verifican que el sistema completo cumple con todos los requisitos especificados. Sus herramientas populares son: Selenium y cypress.

Pruebas de aceptación: Las pruebas de aceptación, también conocidas como pruebas de usuario, se centran en validar que el software cumple con los requisitos del usuario y que este tenga una buena experiencia. Por lo tanto, su

objetivo es asegurar que el software sea aceptado por los usuarios finales y cumpla con los criterios de aceptación. Sus herramientas populares son: Cucumber y behave.

Pruebas de carga: Las pruebas de carga evalúan el rendimiento del software bajo condiciones de carga máxima o elevada a la normal. Lo que se busca básicamente es identificar posibles errores y evaluar la capacidad de respuesta del sistema. Al determinar la capacidad del sistema para manejar la carga prevista, esta ha cumplido su objetivo. Sus herramientas populares son: Apache JMeter y Gatling

Pruebas de estrés: Estas llevan al sistema más allá de sus límites para así evaluar su comportamiento en situaciones extremas. Esto incluye la saturación de recursos o la manipulación de datos. Su objetivo es identificar el punto donde se rompe el sistema y evaluar su capacidad de recuperación después de situaciones de alta demanda. Sus herramientas populares son: Apache JMeter y Gatling.

-Exploración de técnicas (TDD y BDD).

El TDD (Test Driven Development) es una técnica de desarrollo de software en la que las pruebas son escritas antes de escribir el código de producción. El ciclo de desarrollo en TDD sigue una serie de tres pasos los cuales son: escribir una prueba que falle, escribir el código mínimo para pasar la prueba y finalmente, refactorizar el código para mejorar su calidad. Esto hace que mejore la calidad del código desde el principio y proporciona documentación.

Por otro lado, el BDD (Behavior Driven Development) se centra en el comportamiento del software desde la perspectiva del usuario. Utiliza un lenguaje natural y describe el comportamiento del sistema en términos de escenarios y especificaciones comprensibles por los no desarrolladores. Esto facilita la colaboración entre desarrolladores y la comprensión del comportamiento del software.

-La automatización de pruebas.

Introducción:

La automatización de pruebas es un proceso en el que se utilizan herramientas y scripts para ejecutar pruebas de software predefinidas en el código de la aplicación. Este enfoque permite la repetición eficiente de pruebas, acelera el ciclo de desarrollo y garantiza una mayor cantidad de pruebas.

Ventajas:

Esta es muy eficiente, ya que la ejecución automatizada es más rápida y eficiente que las pruebas manuales. La cantidad de repeticiones que se pueden realizar en distintas configuraciones y entornos es otra ventaja, por no mencionar la posibilidad de realizar pruebas exhaustivas y repetitivas sin aumentar significativamente el esfuerzo que haga el desarrollador. También facilita la detección temprana de errores durante el ciclo de desarrollo y ahorra mucho tiempo.

Algunas herramientas y Frameworks populares para la automatización de pruebas pueden ser: Selenium, la cual es una herramienta de código abierto para la automatización de pruebas de aplicaciones web. JUnit, el cual sirve para probar aplicaciones Java, este facilita la creación y ejecución de pruebas unitarias. Cypress es muy usado también, la cual es una herramienta de prueba de extremo a extremo para aplicaciones web modernas, permite realizar pruebas en el navegador. Por último, cabe mencionar a TestNG, el framework de prueba inspirado en JUnit y hecho para pruebas más flexiblesy potentes en Java.

-Casos de uso y ejemplos en proyectos reales.

En el caso de las pruebas unitarias, es muy posible el uso en un sistema de gestión de bibliotecas, en el cual se implementen pruebas unitarias para garantizar que cada función, como la búsqueda de libros por título o autor, devuelva resultados precisos.

En cuanto a las pruebas de integración, podemos llegar a observarlas en un sistema de comercio electrónico, se realizan pruebas de integración para asegurar que los módulos, como pueden ser el carrito de comprar y la pasarela de pago, funcionen sin ningún problema juntos. Un ejemplo es la simulación de una transacción de compra completa, desde agregar productos, hasta realizar el pago.

Un caso de una prueba de sistema puede ser un sistema de reserva de vuelos, donde se llevan a cabo pruebas a dicho sistema para confirmar que todas las acciones, desde la selección del vuelo, hasta la entrega del billete se realicen de forma correcta. Un ejemplo es una prueba de extremo a extremo para simular dicho proceso de reserva.

-Conclusión.

El testing y las pruebas de código son fundamentales para garantizar la calidad del software. Al implementar técnicas como las que he mencionado anteriormente (TDD y BDD) y la automatización de pruebas, los equipos pueden lograr un desarrollo mucho mejor, identificando los problemas mucho más temprano y rápido, manteniendo así la confiabilidad del software. La inversión en pruebas es esencial para ofrecer productos de alta calidad y satisfacer las expectativas de los usuarios.