


PORTADA

Nombre Alumno / DNI	Jorge Madrid <u>Valnickas</u> / 03203272Q
Título del Programa	1ºBD Cybersecurity & Hacking
Nº Unidad y Título	Unit 1 - Programming & Coding
Año académico	2023-2024
Profesor de la unidad	Gabriela García
Título del Assignment	Assignment Brief
Día de emisión	20/01/2024
Día de entrega	31/01/2024
Nombre IV y fecha	
Nombre IV y fecha 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:</p> <p>Firma del alumno:</p> <p></p> <p></p>

Plagio

El plagio es una forma particular de hacer trampa. El plagio debe evitarse a toda costa y los alumnos que infrinjan las reglas, aunque sea inocentemente, pueden ser sancionados. Es su responsabilidad asegurarse de comprender las prácticas de referencia correctas. Como alumno de nivel universitario, se espera que utilice las referencias adecuadas en todo momento y mantenga notas cuidadosamente detalladas de todas sus fuentes de materiales para el material que ha utilizado en su trabajo, incluido cualquier material descargado de Internet. Consulte al profesor de la unidad correspondiente o al tutor del curso si necesita más consejos.

LENGUAJES DE PROGRAMACIÓN Y TESTING DE CÓDIGO

A blurred image of a code editor showing JavaScript code. The code includes an 'else if' statement and several lines of DOM manipulation and validation logic.

```
else if (i==2)
{
  var atpos=inputs[i].indexOf("@");
  var dotpos=inputs[i].lastIndexOf(".");
  if (atpos<1 || dotpos<atpos+2 || dotpos>inputs[i].lastIndexOf(".")-1)
  document.getElementById('errEmail').innerHTML += "Incorrect email format. ";
  else
  document.getElementById(div).innerHTML += "Incorrect email format. ";
}
```

ÍNDICE

1. Lenguajes, paradigmas y estándares de programación:

- a. [Introducción.](#)
- b. [Tipos de Lenguajes de Programación](#)
- c. [Paradigmas de Programación](#)
- d. [Estándares de Programación](#)
- e. [Conclusión](#)
- f. [Referencias](#)

2. Testing y Pruebas de Código:

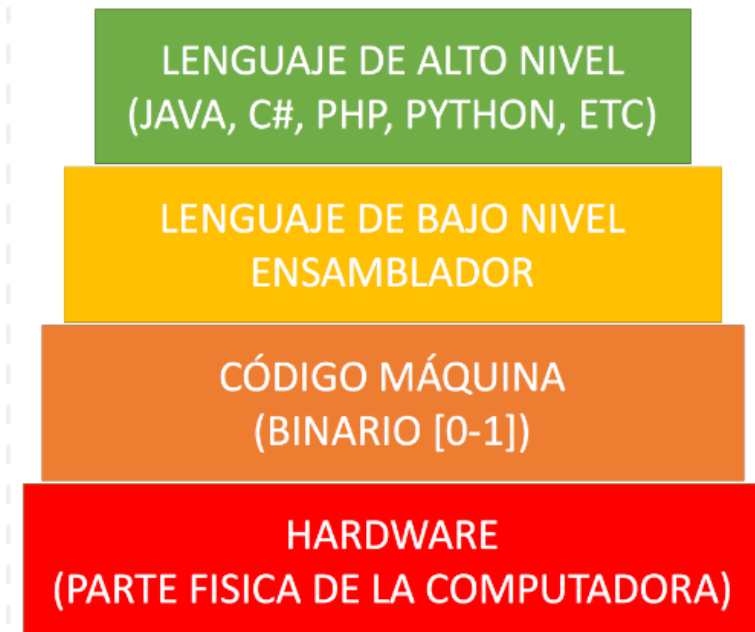
- a. Introducción
- b. Conceptos Básicos
- c. Tipos de Pruebas
- d. Técnicas de Testing
- e. Automatización de Pruebas
- f. Casos de Uso y Ejemplos
- g. Conclusión

I. A INTRODUCCIÓN

Un **lenguaje de programación** es un conjunto de reglas gramaticales (tanto sintácticas como semánticas) que instruyen a que un ordenador o dispositivo se comporte de una cierta manera. Cada lenguaje de programación tiene un vocabulario, un conjunto único de palabras clave que sigue a una sintaxis especial para formar y organizar instrucciones del ordenador.

Programación es el proceso de análisis, diseño, implementación, prueba y depuración de un algoritmo, a partir de un lenguaje que compila y genera un código fuente ejecutado en la computadora.

I. B. TIPOS DE LENGUAJES DE PROGRAMACIÓN

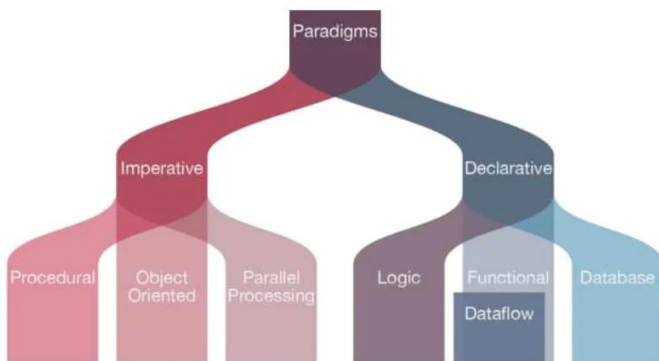


Los tipos de lenguajes de programación se pueden clasificar principalmente como lenguajes de programación de bajo y alto nivel. Aunque son simples en comparación con los lenguajes humanos, los lenguajes de alto nivel son más complejos que los de bajo nivel. Al mismo tiempo, un lenguaje de alto nivel ofrece más legibilidad en comparación con su contraparte de bajo nivel, cuya interpretación necesita un conocimiento especializado en arquitectura informática.

Normalmente se distingue entre los siguientes tipos de lenguaje de programación:

- **Lenguajes 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. Ejemplo de esto sería el lenguaje ensamblador.
- **Lenguajes 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. Ejemplos de estos serían Python y Java
- **Lenguajes de nivel medio.** Este término no siempre es aceptado, 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. Ejemplos de estos serían C y Basic.

I. C. 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.** El paradigma imperativo o de procedimientos es, probablemente, uno de los paradigmas más conocidos en el mundo de la programación. Como su nombre lo indica, este es un método que permite desarrollar programas a través de procedimientos. Mediante una serie de instrucciones, se explica paso por paso cómo funciona el código para que el proceso sea lo más claro posible.
- **Paradigma funcional.** Una de las características del paradigma funcional es que este, como su nombre lo indica, trabaja a través de determinadas funciones matemáticas. Este es un tipo de paradigma que se usa, principalmente, en el ámbito académico más que en el comercial. A diferencia del paradigma imperativo, aquí importa más el “qué” y no tanto el “cómo” se desarrolla un proyecto.
- **Paradigma declarativo.** El paradigma declarativo es aquel que se preocupa por el resultado final desde el inicio. Determinar de forma automática la ruta a seguir para conseguir una solución puede resultar muy eficaz a la hora de programar, solo se necesita tener claridad en torno al proceso que se va a llevar adelante.
- **Paradigma reactivo.** El paradigma reactivo está enfocado en analizar el flujo de datos, ya sean finitos o infinitos, con el fin de responder a las necesidades que se presenten durante el desarrollo de los proyectos en términos de escalado, y para procurar una reacción inmediata al cambio de valores que se producen por los flujos de datos.
- **Paradigmas de programación orientada a objetos.** Este tipo de paradigma de programación ofrece una guía que permite identificar cómo trabajar con él a través de objetos y planos de código. Este tipo de paradigma se constituye por piezas simples u objetos que al relacionarse entre sí forman diferentes componentes del sistema que estemos trabajando.
- **Paradigma lógico.** Es otro paradigma de la programación que existe de manera tradicional pero que no ha llegado a extenderse de manera relevante. Se trata de una programación basada en el cálculo de predicados (una teoría matemática que permite lograr que un ordenador basándose en hecho y reglas lógicas, pueda dar soluciones inteligentes). Se suele utilizar en la inteligencia artificial y pequeños programas infantiles, pero no existen muchos lenguajes que la implementen.

Otra forma de clasificación a menudo es la siguiente:

Lenguajes imperativos. Menos flexibles, dada la secuencialidad en que construyen sus instrucciones, estos lenguajes programan mediante órdenes condicionales y un bloque de comandos al que retornan una vez llevada a cabo la función.

Lenguajes funcionales. También llamados procedimentales, estos lenguajes programan mediante funciones que son invocadas conforme a la entrada recibida, que a su vez son resultado de otras funciones.

Existe otra forma adicional de clasificar a los lenguajes de programación, en la cual hay 2 tipos, lenguajes interpretados y lenguajes compilados:

- Un **lenguaje interpretado** es un lenguaje de programación que está diseñado para ser ejecutado por medio de un intérprete, en contraste con los lenguajes compilados. Cualquier lenguaje puede ser compilado o interpretado, así que esta denominación es aplicada debido a la práctica de funcionamiento común y no a alguna característica subyacente de un lenguaje en particular. Sin embargo, hay lenguajes que son diseñados para ser en concreto interpretativos, por lo tanto un compilador causará una carencia de la eficacia.
- Los **lenguajes de programación compilados** son lenguajes de alto nivel que requieren que las instrucciones (código fuente del programa), sean traducidas, -mediante un programa compilador-, a un lenguaje que entienda la máquina (lenguaje máquina), con el fin de generar una versión ejecutable del programa.

I. D. ESTÁNDARES DE PROGRAMACIÓN

Estilo de programación (también llamado **estándares de código**, guías de estilo o convención de código) es un término que describe convenciones para escribir código fuente en ciertos lenguajes de programación. El estilo de programación es dependiente del lenguaje de programación que se haya elegido para programar, es decir que cada lenguaje puede tener distintas convenciones pero cada lenguajes tiene las propias.

Características de los estándares:

Nombres de variable apropiadas:

Una pieza clave para un buen estilo es la elección apropiada de nombres de variable. Variables pobremente nombradas dificultan la lectura del código fuente y su comprensión. Como ejemplo, considérese el siguiente extracto de pseudocódigo:

Estilo de sangría:

En lenguajes de programación que usan llaves para sangrar o delimitar bloques lógicos de código, como por ejemplo C, es también un punto clave el buen estilo. Usando un estilo lógico y consistente hace el código de uno más legible.

Bucles y estructuras de control:

El uso de estructuras de control lógicas para bucles también es parte de un buen estilo de programación. Ayuda a alguien que esté leyendo el código a entender la secuencia de ejecución (en programación imperativa).

Espaciado:

Los lenguajes de formato libre ignoran frecuentemente los espacios en blanco. El buen uso del espaciado en la disposición del código de uno es, por tanto, considerado un buen estilo de programación.

I. E. CONCLUSIÓN

En conclusión, analizar los tipos de lenguajes de programación, paradigmas de programación y estándares en el desarrollo de software revela la complejidad y riqueza del mundo tecnológico. La diversidad de lenguajes proporciona a los desarrolladores una amplia gama de herramientas para resolver problemas específicos y enfatiza la importancia de elegir sabiamente en función de las necesidades del proyecto. Comprender los paradigmas de programación proporciona una perspectiva conceptual que permite a los equipos adoptar enfoques flexibles y creativos para la resolución de problemas, aumentando su adaptabilidad en entornos dinámicos.

Los estándares se consideran la base para la coherencia y la interoperabilidad en sistemas complejos. Desempeñan un papel importante en el establecimiento de reglas comunes que facilitan la colaboración entre equipos y la integración eficiente de componentes de software. El cumplimiento de las normas no sólo promueve la estabilidad a corto plazo, sino que también contribuye a la sostenibilidad a largo plazo de las soluciones tecnológicas.

En un entorno tecnológico en constante evolución, la capacidad de un desarrollador para comprender y aplicar diferentes lenguajes, paradigmas y estándares es un diferenciador clave. La versatilidad y la adaptabilidad han demostrado ser habilidades esenciales para que los desarrolladores de software no sólo se mantengan al día con las tendencias, sino también para innovar y resolver problemas de manera eficiente. En última instancia, una comprensión profunda de estos fundamentos no sólo impulsará el desarrollo de software, sino que también sentará las bases para un progreso continuo y sostenible en un entorno tecnológico cambiante.

I. F. REFERENCIAS

<https://www.chakray.com/es/lenguajes-programacion-tipos-caracteristicas/>

https://programas.cuaed.unam.mx/repositorio/moodle/pluginfile.php/1023/mod_resource/content/1/contenido/index.html/

<https://concepto.de/lenguaje-de-programacion/>

<https://cognosonline.com/co/blog/que-son-paradigmas-de-programacion/>

<https://desarrolloweb.com/articulos/paradigmas-programacion>

https://departamentos.colegiosansaturio.com/deptomatesweb/4ESO/informatica%20web/temas/Unidad_6/pagina1.html

https://es.wikipedia.org/wiki/Estilo_de_programaci%C3%B3n

2. A. INTRODUCCIÓN

El testing y las pruebas de código son elementos fundamentales en el ciclo de vida del desarrollo de software, desempeñando un papel crucial para garantizar la calidad, fiabilidad y seguridad de las aplicaciones. Son elementos esenciales para garantizar un desarrollo de software exitoso, mejorando la calidad, confiabilidad y mantenibilidad de las aplicaciones, al tiempo que contribuyen a la satisfacción del usuario y la minimización de riesgos.

2. B. CONCEPTOS BÁSICOS

Definición y diferencia entre testing y pruebas de código:

Testing:

El testing, o prueba de software, es un proceso integral que implica la ejecución de un programa o sistema con el propósito de identificar errores y garantizar que el software funcione según lo previsto. Este proceso abarca diversas fases, desde la planificación y diseño de pruebas hasta la ejecución y análisis de resultados. El objetivo principal del testing es asegurar la calidad del software, detectar posibles problemas y garantizar que el producto cumple con los requisitos establecidos.

Pruebas de Código:

Las pruebas de código, por otro lado, se centran específicamente en evaluar la calidad y el comportamiento del código fuente de una aplicación. Estas pruebas se realizan para identificar errores, vulnerabilidades y mejorar la legibilidad del código. Las pruebas de código incluyen diferentes niveles, desde pruebas unitarias que evalúan funciones o métodos individuales hasta pruebas de integración que examinan la interacción entre diferentes unidades de código. El objetivo principal de las pruebas de código es garantizar que el código sea robusto, mantenible y cumpla con los estándares de calidad.

Diferencia:

- Testing (Prueba de Software):
 - Alcance: Aborda el funcionamiento global del software, evaluando aspectos como la funcionalidad, rendimiento, seguridad y usabilidad.
 - Enfoque: Se concentra en verificar si el software cumple con los requisitos y expectativas del usuario final.

- Pruebas de Código:
 - Alcance: Se centra exclusivamente en el código fuente de la aplicación, evaluando su calidad, corrección y eficiencia.
 - Enfoque: Busca identificar errores a nivel de código y mejorar la estructura y legibilidad de este.

Objetivos y beneficios de realizar pruebas:

- Objetivos de las Pruebas de Código:
 - Identificar Errores: El objetivo principal es encontrar y corregir errores en el código fuente antes de que impacten en el funcionamiento general de la aplicación.
 - Mejorar la Legibilidad: Las pruebas de código también buscan mejorar la legibilidad y mantenibilidad del código, asegurando que sea comprensible y fácil de mantener para los desarrolladores.
 - Asegurar la Correcta Implementación: Verificar que el código implemente correctamente la lógica y la funcionalidad prevista, cumpliendo con los requisitos establecidos.
 - Optimizar el Rendimiento: Identificar áreas del código que podrían ser optimizadas para mejorar el rendimiento de la aplicación.
 - Garantizar la Seguridad: Evaluar la seguridad del código para prevenir vulnerabilidades y riesgos de seguridad.
 - Facilitar el Refactoring: Proporcionar una red de seguridad al realizar cambios en el código, asegurando que las modificaciones no introduzcan errores y manteniendo la funcionalidad esperada.
- Objetivos de las Pruebas de Software (Testing):
 - Validar la Funcionalidad: Confirmar que el software cumple con los requisitos y expectativas del usuario, verificando la funcionalidad.
 - Identificar Defectos: Descubrir y corregir defectos, errores o problemas que puedan afectar el rendimiento o la seguridad del software.
 - Asegurar la Calidad: Garantizar la calidad general del software mediante la evaluación de su rendimiento, usabilidad, eficiencia y otros atributos.
 - Cumplir con Estándares y Requisitos: Verificar que el software cumple con estándares, regulaciones y requisitos específicos establecidos para su desarrollo y uso.
 - Optimizar el Rendimiento: Evaluar y mejorar el rendimiento del software para garantizar una experiencia eficiente y satisfactoria para el usuario.

2. C. TIPOS DE PRUEBAS

“Hay muchos tipos diferentes de pruebas de software, cada una con objetivos y estrategias específicos:

- **Prueba de aceptación:** verifica si todo el sistema funciona según lo previsto.
- **Pruebas de integración:** asegura que los componentes o funciones del software operen juntos.
- **Pruebas de unidad:** valida que cada unidad de software funcione según lo esperado. Una unidad es el componente de prueba más pequeño de una aplicación.
- **Pruebas funcionales:** verifica funciones mediante la emulación de escenarios de negocio, en función de los requisitos funcionales. La prueba de caja negra es una forma común de verificar funciones.
- **Pruebas de rendimiento:** prueba cómo funciona el software bajo diferentes cargas de trabajo. Las pruebas de carga, por ejemplo, se utilizan para evaluar el rendimiento en condiciones de carga reales.
- **Pruebas de regresión:** verifica si las nuevas características rompen o degradan la funcionalidad. Las pruebas de cordura se pueden utilizar para verificar menús, funciones y comandos a nivel superficial, cuando no hay tiempo para una prueba de regresión completa.
- **Pruebas de estrés:** prueba cuánta tensión puede soportar el sistema antes de que falle. Considerado como un tipo de prueba no funcional.
- **Pruebas de usabilidad:** valida qué tan bien un cliente puede usar un sistema o una aplicación web para completar una tarea.”

(IBM, s.f.)

2. D. TIPOS DE TESTING

“TDD o Test-Driven Development (desarrollo dirigido por tests) es una práctica de programación que consiste en escribir primero las pruebas (generalmente unitarias), después escribir el código fuente que pase la prueba satisfactoriamente y, por último, refactorizar el código escrito.” (Herranz, 2023)

BDD o Behavior Driven Development es un proceso de desarrollo de software nacido desde el desarrollo guiado por pruebas. El BDD es una práctica derivada del Test-Driven Development (TDD), que también se guía por el enfoque Specification by Example (SBE). Dicho método define los requisitos y pruebas funcionales de la aplicación por medio de ejemplos realistas y no abstractos.

Debido al análisis brindado por BDD sobre el comportamiento del software en desarrollo, la documentación creada también se ve beneficiada.

(TECNOVA, s.f.)



2. E. AUTOMATIZACIÓN DE PRUEBAS.

- TDD (Desarrollo Guiado por Pruebas) y BDD (Desarrollo Guiado por Comportamiento) son dos enfoques populares en el desarrollo de software, pero también existen otros conceptos y prácticas importantes relacionados con las pruebas y la calidad del código. Aquí hay algunos términos adicionales que pueden ser relevantes:
- ATDD (Desarrollo Guiado por Aceptación):
 - Descripción: Similar a BDD, ATDD se centra en la colaboración entre desarrolladores, testers y partes interesadas para definir y validar requisitos a través de casos de prueba que representan comportamientos aceptables del sistema.
- Test Unitario (Unit Test):
 - Descripción: Una prueba unitaria es una prueba automatizada que se enfoca en verificar la funcionalidad de una unidad de código individual, como una función o método.
- Pruebas Funcionales Automatizadas:
 - Descripción: Incluyen pruebas automatizadas que verifican las funciones y características del software desde una perspectiva del usuario final.
- Pruebas de Integración Continua (CI):
 - Descripción: La práctica de integrar automáticamente cambios de código en un repositorio compartido y ejecutar pruebas automatizadas de forma continua para detectar problemas de integración de manera temprana.
- Pruebas de Regresión Automatizadas:
 - Descripción: Conjunto de pruebas automatizadas diseñadas para detectar regresiones, es decir, garantizar que los cambios recientes no hayan introducido errores en funcionalidades existentes.
- Pruebas de Unidad Funcional (Functional Unit Testing):
 - Descripción: Pruebas que evalúan el comportamiento funcional de una unidad de código, centrándose en la entrada y salida de funciones o métodos.
- Pruebas de Humo (Smoke Testing):
 - Descripción: Un tipo de prueba rápida y básica que verifica si las funciones principales del software funcionan correctamente después de cambios significativos.
- Pruebas de Estrés (Stress Testing):
 - Descripción: Evalúan cómo el sistema se comporta bajo cargas de trabajo extremas o condiciones de estrés.
- Pruebas de Penetración (Penetration Testing):
 - Descripción: Pruebas diseñadas para identificar vulnerabilidades de seguridad en el software, simulando ataques reales.
- Pruebas Exploratorias:
 - Descripción: Pruebas realizadas de manera ad hoc sin un plan predefinido, explorando el software para identificar posibles problemas.

2. F CASOS DE USO Y EJEMPLOS

Yo he hecho pruebas de aceptación regresión y de la usabilidad: Ya que haciendo el carrito de compra para mi página web, no me funcionaba, y no sabía si era alguna clase en el html que no había puesto o que el script de JavaScript estaba mal escrito.

Esto lo arreglé fijándome en todas las clases de HTML y los errores de la consola en el navegador

2. G. CONCLUSIÓN

En resumen, las pruebas de software y de código son procesos esenciales para garantizar la calidad y confiabilidad de las aplicaciones. Mientras que las pruebas de software abordan la funcionalidad global, las pruebas de código se centran en la evaluación específica del código fuente. Ambas prácticas son cruciales para la identificación temprana de errores, la mejora continua y el éxito en el desarrollo de software.

2. H. REFERENCIAS

Herranz, J. I., 2023. *paradigmadigital.com*. [En línea]

Available at: <https://www.paradigmadigital.com/dev/tdd-como-metodologia-de-diseno-de-software/>

[Último acceso: 31 Enero 2024].

IBM, s.f. *IBM*. [En línea]

Available at: <https://www.ibm.com/mx-es/topics/software-testing>