


PORTADA

Nombre Alumno / DNI	Gadea Diez Santamria-45577194A
Título del Programa	Unit1 Programming & Coding
Nº Unidad y Título	UNIT 1
Año académico	2023-2024
Profesor de la unidad	Gabriela García
Título del Assignment	AB FINAL
Día de emisión	13/10/2023
Día de entrega	31/01/2024
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: 31/01/2024</p> <p>Firma del alumno:</p> 

Plagio

El plagio es una forma particular de hacer trampa. El plagio debe evitarse a todo costo 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.

INTRODUCCIÓN

Antes de continuar con este informe vamos a responder a una pregunta que es básica para entender de qué va este trabajo.

¿Qué es la programación?

Con esta pregunta podemos decir que es planificar, por ejemplo, un despertador para que suene a las seis de la mañana todos los días menos los sábados y los domingos. La respuesta no es incorrecta, pero adentrándonos en el tema que nos incumbe. La programación de software es el proceso de crear conjuntos de instrucciones, llamados código fuente, que indican a una computadora cómo realizar una tarea específica.

Cuáles son los pasos de la programación:

Definición del problema: Comprender el problema que se desea resolver y determinar los requisitos del software.

Diseño: Planificar la estructura y la lógica del software, identificando los componentes necesarios y cómo interactúan entre sí.

Codificación: Escribir el código fuente utilizando un lenguaje de programación. Aquí es donde se traducen los conceptos y el diseño en instrucciones que una computadora puede entender.

Pruebas: Probar el software para identificar y corregir errores (bugs) y asegurarse de que funcione según lo previsto.

Depuración: Corregir errores y realizar ajustes en el código para mejorar el rendimiento y la eficiencia.

Implementación: Poner en marcha el software y hacerlo disponible para su uso.

Mantenimiento: Realizar actualizaciones y correcciones a lo largo del tiempo para adaptarse a cambios en los requisitos o corregir posibles problemas.

Para cerrar esta introducción, veamos unos datos curiosos:

El primer programador en todo el mundo fue una mujer. Se llamaba Ada Lovelace y nació en 1815. Algunas personas creen que Lovelace publicó el primer algoritmo destinado a ser ejecutado por una máquina.

El primer lenguaje de programación del mundo se llamó FORTRAN (acrónimo de Formula Translation), y fue creado en 1956.

El primer «bug de la computadora» fue llamado así a causa de un insecto real. Este bicho fue encontrado por Grace Hopper. El ordenador que ella estaba construyendo comenzó a fallar y fue entonces cuando descubrió una polilla de verdad en el sistema. Desde entonces, cuando algo falla en el software o en el hardware, lo llamamos «bug»

Tipos de Lenguajes de Programación

▣ **Lenguaje alto nivel:** Se caracteriza por expresar los algoritmos de una manera adecuada a la capacidad cognitiva humana, en lugar de la capacidad con que las máquinas lo ejecutan. Estos lenguajes permiten una máxima flexibilidad al programador a la hora de abstraerse o de ser literal. Ejemplo: Python, Ruby, Java Script.

▣ **Lenguaje medio nivel:** Son precisos para ciertas aplicaciones como la creación de sistemas operativos, ya que permiten un manejo abstracto (independiente de la máquina, a diferencia del ensamblador), pero sin perder mucho del poder y eficiencia que tienen los lenguajes de bajo nivel. Ejemplo C++, Java.

Una característica distintiva, por ejemplo, que convierte a C en un lenguaje de medio nivel y al Pascal en un lenguaje de alto nivel es que en el primero es posible manejar las letras como si fueran números (en Pascal no), y por el contrario en Pascal es posible concatenar las cadenas de caracteres con el operador suma y copiarlas con la asignación (en C es el usuario el responsable de llamar a las funciones correspondientes).

▣ **Lenguaje de medio nivel:** Son aquellos que, basándose en los juegos de instrucciones disponibles (chip set), permiten el uso de funciones a nivel aritmético, pero a nivel lógico dependen de literales en ensamblador. Estos lenguajes están orientados a procedimientos.

▣ **Lenguaje bajo nivel o ensamblador:** Los lenguajes de bajo nivel, también llamados lenguajes ensambladores, permiten al programador escribir instrucciones de un programa usando abreviaturas del inglés, también llamadas palabras nemotécnicas, tales como: ADD, DIV, SUB, etc. Un programa escrito en un lenguaje ensamblador tiene el inconveniente de que no es

comprensible para la computadora, ya que, no está compuesto por ceros y unos. Para traducir las instrucciones de un programa escrito en un lenguaje ensamblador a instrucciones de un lenguaje máquina hay que utilizar un programa llamado ensamblador. Ejemplo: C, Perl, COBOL, mov.

Abstracción: Una abstracción es una manera de reducir la complejidad y permitir un diseño e implementación más eficiente en sistemas de software complejos.

Ventajas de la abstracción:

- ☐ Ayuda al usuario a evitar escribir código de bajo nivel.
- ☐ Evitar duplicar códigos y aumenta la reusabilidad.
- ☐ Se puede cambiar la implementación interna de la clase de forma independiente sin afectar al usuario.
- ☐ Ayuda a aumentar la seguridad de la aplicación o programa ya que solo los detalles importantes son proporcionados al usuario.

Python: Analítica y procesamiento de datos.. Por ejemplo: Pinterest, Panda 3D, Dropbox, Spotify, Netflix, Uber, Instagram, Reddit, Google, YouTube.

Ruby: Permite desarrollar distintos tipos de aplicaciones.

Aplicaciones de servicio web, clientes de correo electrónico, procesamiento de datos y aplicaciones de red. Entre ellas, las más conocidas son Twitter, Twitch, Groupon, Airbnb y Shopify.

JavaScript: Es un lenguaje de programación que los desarrolladores utilizan para hacer páginas web interactivas. Yahoo, Wikipedia, Wikipedia están programadas con este lenguaje.

Perl: Es un lenguaje de programación de propósito general desarrollado originalmente para la manipulación de texto y ahora se usa para una amplia gama de tareas, incluida la administración de sistemas, desarrollo web, programación de redes, desarrollo de GUI

PARADIGMA DE PROGRAMACIÓN

Paradigma de programación imperativo: Es uno de los paradigmas de programación más antiguos. Presenta una estrecha relación con la arquitectura de la máquina. Está basado en la arquitectura de Von Neumann. Funciona cambiando el estado del programa mediante declaraciones de asignación. Realiza tareas paso a paso cambiando de estado. Ejemplo: FORTRAN, Algol, Pascal, C, Modula-2, Ada.

Ventajas:

- o Muy simple de implementar.
- o Contiene bucles, variables, etc.

Desventajas:

- El problema complejo no se puede resolver.
- Menos eficiente y productivo.
- La programación paralela no es posible.

Paradigma de programación declarativo: Este paradigma no necesita definir algoritmos puesto que describe el problema en lugar de encontrar una solución al mismo. Este paradigma utiliza el principio del razonamiento lógico para responder a las preguntas o cuestiones consultadas.

Este paradigma a su vez se divide en dos:

Programación Lógica: Prolog.

Programación funcional: Lisp, Scala, Java, Kotlin.

Programación orientada a objetos: En este modelo de paradigma se construyen modelos de objetos que representan elementos (objetos) del problema a resolver, que tienen características y funciones. Permite separar los diferentes componentes de un programa, simplificando así su creación, depuración y posteriores mejoras. La programación orientada a objetos disminuye los errores y promueve la reutilización del código. Es una manera especial de programar, que se acerca de alguna manera a cómo expresaríamos las cosas en la vida real. Ejemplos de lenguajes de programación orientados a objetos serían Java, Python o C#.

La programación funcional: Un lenguaje de programación declarativo donde el programador especifica lo que quiere hacer, en lugar de lidiar con el estado de los objetos. Es decir, las funciones estarían en un primer lugar y nos centraremos en expresiones que pueden ser asignadas a cualquier variable. Ejemplos: Perl, Ruby, JavaScript, Python.

El paradigma lógico: Denominado también como programación predicativa, se basa en la lógica matemática. En lugar de una sucesión de instrucciones, un software programado según este principio contiene un conjunto de principios que se pueden entender como una recopilación de hechos y suposiciones. Todas las solicitudes al programa se procesan de forma que el intérprete recurre a estos principios y les aplica reglas definidas previamente para alcanzar el resultado deseado. Lenguaje: Prolog.

ESTÁNDARES DE PROGRAMACIÓN

Los estándares de código son una serie de reglas definidas para un lenguaje de programación, o bien, un estilo de programación específico. El estilo garantiza que todos los ingenieros que contribuyen a un proyecto tengan una forma única de diseñar su código, lo que da como resultado una base de código coherente, asegurando una fácil lectura y mantenimiento. Cada lenguaje tiene sus propios estándares.

Los más populares: Variables (facilidad de lectura y comprensión), Indentación (más legible), claridad, inclusión (explicar funciones).

BENEFICIOS DE ADHERIRSE

Mantener la web gratis y accesible para todos

Ayudar a simplificar el código fuente

Reducción del tiempo de desarrollo y mantenimiento

Hacer de la web un lugar más accesible

Permitir compatibilidad y validación hacia atrás

Ayudar a mantener una mejor optimización de motores de búsqueda

Crear un grupo de conocimiento común

Introducción:

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 de software, es un proceso para verificar y validar la funcionalidad de un programa o una aplicación de software con el objetivo de garantizar que el producto de software esté libre de defectos. En el mundo del desarrollo de software se trata de probar que una pieza de nuestro código funciona correctamente.

Aunque testing y pruebas de código tiene ciertos parecido también hay diferencias. Veamos las más llamativas.

Testing:

- Abarca más terreno: Acoge todo el proceso para asegurar la calidad del software.
- Incluye pruebas manuales y automáticas: Pueden hacer pruebas tanto humanos como máquinas.
- Enfoque en la funcionalidad: Su objetivo es que el software funcione y cumpla con los requisitos.

Pruebas de Código:

- Enfoque más específico: Solo se centra en la calidad y el funcionamiento del código fuente.

- Pruebas automáticas: A diferencia del testing que las tiene tanto manuales como automáticas las de prueba de código suelen ser automáticas.
- Objetivo: El objetivo es encontrar errores en el código.

Objetivos y beneficios de realizar pruebas.

Objetivos:

El objetivo principal del testing es asegurar que el software cumpla con los requisitos especificados y que funcione correctamente.

- Corregir errores: Identificar y corregir errores o defectos en el software es uno de los objetivos clave. Cuanto antes se detecten y se corrijan mejor.
- Validar funciones: Las funciones puedan ser usadas por un usuario.

Beneficios:

- Mejorar la calidad del software: Teniendo ya los errores identificados la calidad del software mejora.
- Código más robusto y confiable: Las pruebas de código ayudan a fortalecer y mejorar la calidad del código, lo que resulta en un software más robusto y confiable.
- Mejora la comprensión del código: Las pruebas de código sirven para comprender mejor el código y que les sea más sencillo a los programadores.
- Facilita el mantenimiento del código: Un código bien probado es más fácil de mantener, ya que se pueden realizar cambios y actualizaciones con menor riesgo de introducir nuevos errores.

TIPOS DE PRUEBAS

1. Pruebas unitarias

Las pruebas unitarias son de muy bajo nivel y se realizan cerca de la fuente de la aplicación. Consisten en probar métodos y funciones individuales de las clases, componentes o módulos que usa tu software. En general, las pruebas unitarias son bastante baratas de automatizar y se pueden ejecutar rápidamente mediante un servidor de integración continua.

Herramientas Comunes: JUnit, NUnit, PyTest.

2. Pruebas de integración

Las pruebas de integración verifican que los distintos módulos o servicios utilizados por tu aplicación funcionan bien en conjunto. Por ejemplo, se puede probar la interacción con la base de datos o asegurarse de que los microservicios funcionan bien en conjunto y según lo esperado. Estos tipos de pruebas son más costosos de ejecutar, ya que requieren que varias partes de la aplicación estén en marcha.

Herramientas: TestNG, Mocha, PHPUnit.

3. Pruebas funcionales

Las pruebas funcionales se centran en los requisitos empresariales de una aplicación. Solo verifican el resultado de una acción y no comprueban los estados intermedios del sistema al realizar dicha acción.

A veces, se confunden las pruebas de integración con las funcionales, ya que ambas requieren que varios componentes interactúen entre sí. La diferencia es que una prueba de integración puede simplemente verificar que puedes hacer consultas en la base de datos, mientras que una prueba funcional esperaría obtener un valor específico desde la base de datos, según dicten los requisitos del producto.

Herramientas: Selenium, laboratorios de salsa.

4. Pruebas de extremo a extremo

Las pruebas integrales replican el comportamiento de un usuario con el software en un entorno de aplicación completo. Además, verifican que diversos flujos de usuario funcionen según lo previsto, y pueden ser tan sencillos como cargar una página web o iniciar sesión, o mucho más complejos, como la verificación de notificaciones de correo electrónico, pagos en línea, etc.

Las pruebas integrales son muy útiles, pero son costosas de llevar a cabo y pueden resultar difíciles de mantener cuando están automatizadas. Se recomienda tener algunas pruebas

integrales clave y depender más de pruebas de menor nivel (unitarias y de integración) para poder detectar rápidamente nuevos cambios.

5. Pruebas de aceptación

Las pruebas de aceptación son pruebas formales que verifican si un sistema satisface los requisitos empresariales. Requieren que se esté ejecutando toda la aplicación durante las pruebas y se centran en replicar las conductas de los usuarios. Sin embargo, también pueden ir más allá y medir el rendimiento del sistema y rechazar cambios si no se han cumplido determinados objetivos.

Herramientas: :** Cucumber, Behave, SpecFlow.

6. Pruebas de rendimiento

Las pruebas de rendimiento evalúan el rendimiento de un sistema con una carga de trabajo determinada. Ayudan a medir la fiabilidad, la velocidad, la escalabilidad y la capacidad de respuesta de una aplicación. Por ejemplo, una prueba de rendimiento puede analizar los tiempos de respuesta al ejecutar un gran número de solicitudes, o cómo se comporta el sistema con una cantidad significativa de datos. Puede determinar si una aplicación cumple con los requisitos de rendimiento, localizar cuellos de botella, medir la estabilidad durante los picos de tráfico y mucho más.

7. Pruebas de humo

Las pruebas de humo son pruebas básicas que sirven para comprobar el funcionamiento básico de la aplicación. Están concebidas para ejecutarse rápidamente, y su objetivo es ofrecerte la seguridad de que las principales funciones de tu sistema funcionan según lo previsto.

Las pruebas de humo pueden resultar útiles justo después de realizar una compilación nueva para decidir si se pueden ejecutar o no pruebas más caras, o inmediatamente después de una implementación para asegurarse de que la aplicación funciona correctamente en el entorno que se acaba de implementar.

8. Pruebas de Carga

Evaluar el rendimiento del sistema bajo condiciones de carga máxima o esperada. Verificar la capacidad del sistema para manejar un volumen significativo de transacciones o usuarios.

Herramientas: Apache JMeter, LoadRunner, Gatling

HERRAMIENTAS POPULARES ASOCIADAS A CADA TIPO DE PRUEBA

1. TDD (Test Driven Development): El test (TDD) o en español desarrollo guiado por pruebas, es un enfoque de programación que se utiliza durante el desarrollo de software en el que se realizan pruebas unitarias antes de escribir el código.

Beneficios:

Detección temprana de errores: Con su método de escribir pruebas antes de escribir el código facilita identificar errores y problemas de diseño desde el principio.

Facilita el diseño modular: Fomenta la creación de código modular y componentes más pequeños, ya que las pruebas se centran en unidades individuales de funcionalidad.

Retos:

Curva de aprendizaje: Puede haber una curva de aprendizaje para los desarrolladores nuevos en TDD, ya que requiere un cambio en la forma de pensar y abordar el desarrollo.

2. BDD (Behavior Driven Development): BDD es Behavior Driven Development, o lo que es lo mismo en español, desarrollo guiado por comportamiento. Es un proceso de software ágil que busca la colaboración y entendimiento entre desarrolladores, gestores de proyecto y equipo de negocio. Es decir, es el camino para tomar antes de la fase de testing de un proyecto.

Beneficios:

Colaboración mejorada: BDD fomenta la colaboración entre equipos al utilizar un lenguaje común (Gherkin) para escribir especificaciones que son comprensibles tanto por técnicos como por no técnicos.

Retos:

Necesidad de herramientas específicas: BDD a menudo requiere el uso de herramientas específicas para la escritura y ejecución de pruebas, lo que puede requerir tiempo para la adopción.

Automatización de Pruebas:

Las pruebas automatizadas consisten en la aplicación de herramientas de software para automatizar el proceso manual de revisión y validación de un producto de software que lleva a cabo una persona. Ahora, la mayoría de los proyectos de software ágiles y de DevOps

modernos incluyen pruebas automatizadas desde el principio; sin embargo, para apreciar plenamente el valor de dichas pruebas, hay que saber cómo era la vida antes de que se adoptaran de forma generalizada.

1. Selenium: Es un entorno de pruebas de software para aplicaciones basadas en la web. Selenium provee una herramienta de grabar/reproducir para crear pruebas sin usar un lenguaje de scripting para pruebas.

Propósito: facilita la labor de obtener juegos de pruebas para aplicaciones web.

Lenguajes de Programación: Java, C#, Python, Ruby, JavaScript.

2. Appium: Es una herramienta de automatización de código abierto para ejecutar scripts y probar aplicaciones nativas, aplicaciones web y aplicaciones híbridas sobre Android o iOS utilizando un webdriver.

Propósito: Automatización de pruebas de aplicaciones móviles (iOS, Android).

Lenguajes de Programación: Java, C#, Python, Ruby, JavaScript.



3. Cucumber:

Propósito: BDD (Behavior Driven Development) para colaboración entre equipos no técnicos y técnicos.

Lenguajes de Programación Soportados: Java, C#, Ruby, JavaScript.

4. Cypress: Es una herramienta de automatización de pruebas frontend para pruebas de regresión de aplicaciones web. Cypress se ejecuta en Windows, Linux y macOS. La aplicación Cypress es un software de código abierto lanzado bajo la licencia MIT, mientras que Cypress Cloud es una aplicación web.

Propósito: Abordar los puntos débiles que enfrentan los desarrolladores o los ingenieros de control de calidad al probar una aplicación.

Lenguajes de Programación: Javascript

Presenta algunos ejemplos prácticos o casos de uso donde se aplican distintos tipos de pruebas y técnicas de testing en proyectos reales.

Cuando estaba realizando pruebas para ver si mi página era responsive vi que el mayor problema era la barra de navegación. Al querer adaptar mi página a diferentes dispositivos mi barra de navegación se movía entera así que para solucionar este problema decidí usar Bootstrap. Gracias a esta aplicación mi página es responsive y puedo abrirla desde cualquier dispositivo sin ningún tipo de problema.

CONCLUSIÓN

Las pruebas y el testing son un papel fundamental para garantizar la calidad del software coma y dando igual a que proyecto de desarrollo nos estemos refiriendo. Gracias a realizar pruebas podemos identificar errores de forma temprana lo que hace que este fallo no llegó muy lejos.

También mejora la calidad del software porque valida que cumple con los requisitos y lo que quiere el usuario es decir las expectativas que tiene.

Otro punto es que garantiza la funcionalidad es decir hace que la experiencia del usuario sea positiva. En resumen, el testing hace que se cumplan los 3 pilares más importantes que son la calidad la funcionalidad y la seguridad

Los distintos tipos de pruebas en software | Atlassian. (s.f.). Atlassian.

<https://www.atlassian.com/es/continuous-delivery/software-testing/types-of-software-testing>