PORTADA

Nombre Alumno / DNI	Rafael Matarranz Yanes 51511223D
Título del Programa	1ºPHE CYBERSECURITY & DIGITAL INTELLIGENCE
Nº Unidad y Título	UNIT 1-PROGRAMMING & CODING
Año académico	2023/2024
Profesor de la unidad	Gabriela Garcia
Título del Assignment	Assigment Brief
Día de emisión	13/10/2023
Día de entrega	31/01/2024
Nombre IV y fecha	
Declaración del estudiante	**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.** **Fecha:** **Firma del alumno:** 

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.

MSMK University

Matarranz_Rafael_ProyectoFinal

Lenguajes, Paradigmas y Estándares de Programación

Introducción.

Los lenguajes de programación desempeñan una parte crucial en la programación, haciendo un software y eficiente y sostenible. Aquí dejo una breve descripción de la importancia de cada uno:

1. Lenguajes de programación

    - Los lenguajes de programación son cruciales a la hora de que el programador se comunique con el ordenador para mandarle unas tareas o comandos para indicarle como quiere que realice dichas tareas.
    - La elección correcta del lenguaje depende del trabajo que tengamos que hacer ya que esto afectara a la eficiencia, legibilidad del código, mantenibilidad y escalabilidad del software.
    - Cada lenguaje esta diseñado para cumplir diferente función, por eso es importante seleccionar el más adecuado para la tarea.

2. Paradigmas de programación

    - Los paradigmas ayudan a enfocar o estilizar la guía de la estructura y la organización del código. Algunos ejemplos de esto es la programación imperativa, orientada a objetos, funcional y lógica.
    - La selección del paradigma afecta a la forma de la que se solucionan los problemas, el modularidad del código y la facilidad de mantenimiento.
    - Comprender estos paradigmas es importante para un programador ya que le ayudara a ser mas flexible a la hora de abordar nuevos problemas y tener una solución más rápida y efectiva.

3. Estándares de programación

    - Los estándares son pautas aceptadas y normalizadas por la comunidad de programadores para garantizar una calidad de código y una consistencia
    - Los estándares ayudan a la colaboración entre desarrolladores, mejoran la legibilidad y ayuda a mantener una coherencia en el código.

Tipos de lenguajes de programación

Los lenguajes de bajo nivel (*lenguajes ensambladores)*, permiten al programador escribir instrucciones de un programa usando abreviaturas del inglés como: ADD, DIV, SUB, etc. Un programa escrito en un lenguaje ensamblador 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, como se muestra continuación:


```
[0x00000000]> pd
            0x00000000    90              nop
            0x00000001    90              nop
            0x00000002    6800009c00      push 0x9c0000 ; 0x009c0000
            0x00000007    e8c7ace37b      call 0x7be3acd3
               0x7be3acd3(unk)
            0x0000000c    bb04009c00      mov ebx, 0x9c0004
            0x00000011    8903            mov [ebx], eax
            0x00000013    e81903f47b      call 0x7bf40331
               0x7bf40331()
            0x00000018    bb08009c00      mov ebx, 0x9c0008
            0x0000001d    8903            mov [ebx], eax
            0x0000001f    bb00009c00      mov ebx, 0x9c0000
            0x00000024    c60300          mov byte [ebx], 0x0
        .-> 0x00000027    68e8030000      push 0x3e8 ; 0x000003e8
        |   0x0000002c    e81124e37b      call 0x7be32442
        |      0x7be32442(unk)
        `=< 0x00000031    ebf4            jmp 0x100000027
            0x00000033    90              nop
            0x00000034    ff              invalid
            0x00000035    ff              invalid
            0x00000036    ff              invalid
            0x00000037    ff              invalid
```

Algunos de los usos mas comunes del lenguaje de ensamblador es la programación de sistemas operativos, desarrollo de controladores de dispositivos, programación de tiempo real, optimización de código crítico y programación de interrupciones y excepciones.

Lenguaje de medio nivel es un lenguaje de programación que se encuentran entre los lenguajes de alto nivel y los lenguajes de bajo nivel.

Suelen incluirse en los lenguajes de alto nivel, pero permiten ciertos manejos de bajo nivel. Son buenos a la hora de la creación de sistemas operativos, ya que permiten un manejo más, pero sin perder eficiencia que tienen los lenguajes de bajo nivel.

ejemplo C: Hola Mundo!

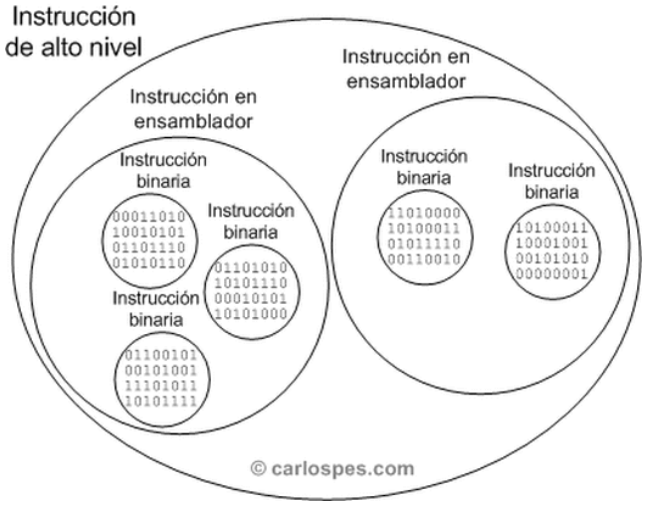
```c
#include <stdio.h>

int main()
{
  printf("Hola Mundo!\n");
  return 0;
}
```

Uno de sus mayores usos es la implementación de algoritmos como listas ligadas, tablas hash y algoritmos de búsqueda y ordenamiento que para otros lenguajes de programación.

El lenguaje de alto nivel es mas parecido a un lenguaje humano, haciéndolo más sencillo y alejándose de lo binario y lo complejo haciéndolo un lenguaje más simple que el resto.

Su principal objetivo es usar un mismo lenguaje para todas las computadoras usando el mismo lenguaje implementando el uso de un traductor o un compilador.

El uso de palabras más parecidas a lo humano hace que el desarrollador se centre más en el código y menos en el lenguaje haciendo más efectivo el código y a la hora de colaborar es más sencillo ya q es más fácil de comprender.




```
def add5(x):
   return x+5

def dotwrite(ast):
   nodename = getNodename()
   label=symbol.sym_name.get(int(ast[0]),ast[0])
   print '    %s [label="%s' % (nodename, label),
   if isinstance(ast[1], str):
      if ast[1].strip():
         print '= %s"];' % ast[1]
      else:
         print '"]'
   else:
      print '"];'
      children = []
      for n, child in enumerate(ast[1:]):
         children.append(dotwrite(child))
      print '    %s -> {' % nodename,
      for name in children:
         print '%s' % name,
```

Los usos más comunes del lenguaje de alto nivel son para desarrollo de aplicaciones software, programación orientada a objetos, desarrollo web, IA, machine learning, automatización de tareas y muchas más.

Paradigmas de programación.

Un paradigma de programación es un enfoque o estilo básico de desarrollo de software. Define cómo se estructuran y organizan las tareas para construir un programa. Es una forma de abordar y resolver problemas a través de la programación. Cada paradigma tiene su propia forma de presentar la información y las acciones que se pueden realizar con ella. A continuación hare un resumen básico de los paradigmas de programación comunes:

1.Programación obligatoria (o procedimental):

Se basa en dar instrucciones al ordenador sobre cómo realizar una tarea paso a paso. Se centra en describir como hacer las cosas. Algunos ejemplos son: C, Pascal, Fortran. Unas características únicas de este paradigma es la modificación de estado, las estructuras de control, los procedimientos/Funciones y el enfoque en el paso a paso.

2.Programación Orientada a Objetos (POO):

Organiza el software en una colección de objetos que contienen tanto datos como comportamiento. Introduce conceptos como herencia, encapsulación y polimorfismo. Algunos ejemplos son: Java, C++, Python, Ruby. Unas características únicas de este paradigma es la organización del código en clase y objetos, la encapsulación, herencia, polimorfismo y un modelado del mundo real mediante abstracción de entidades y sus interacciones.

3.Programación funcional:

Se basa en la evaluación de funciones matemáticas y enfatiza la inmutabilidad (evitación de un estado cambiante). Los programas se tratan como evaluaciones de actividades combinadas. Algunos ejemplos son: Haskell, Lisp, Erlang, Scala. Unas características únicas de este paradigma es el centrado de funciones matemáticas y evitar el cambio de estado, el movimiento de funciones y la inmutabilidad de los datos evitando así cambios en ellos.

4.Programación lógica:

Los programas son vistos como un conjunto de declaraciones y reglas lógicas. Se utiliza principalmente para inferencia y consulta de datos. Algunos ejemplos son: Prólogo. Unas características únicas de este paradigma son sus reglas lógicas y relaciones y su interfaz lógica para reducir resultados.

5.Programación Declarativa:

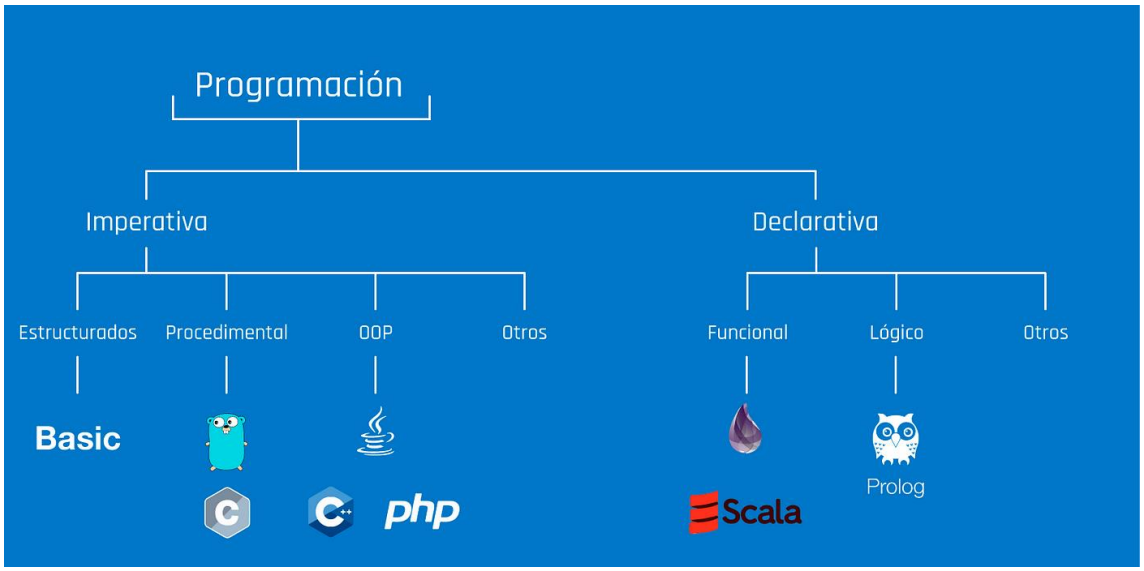
En lugar de describir "cómo" lograr algo (como en la programación imperativa), describe "qué" desea lograr. SQL (utilizado para consultas de bases de datos) es un ejemplo de lenguaje declarativo. Unas características únicas de este paradigma se centra en conseguir los resultados no en como obtenerlos y menos énfasis en el control de flujo explicito.

6.Programación impulsada por eventos (o basada en eventos):

Se centra en responder a eventos como entradas del usuario o señales del sistema. Amplio en el desarrollo de interfaces gráficas de usuario y aplicaciones web. Algunos ejemplos son: JavaScript. Unas características únicas de este paradigma es el uso de eventos como puntos de control, un manejo asincrónico, implementación de listeners, desacoplamiento, una interfaz de usuario interactiva y una escucha continua.

7.Programación paralela y concurrente:

Diseñado para la ejecución simultánea de tareas. Se centra en la creación de software que pueda realizar múltiples funciones simultáneamente. Algunos ejemplos son: Ir, Erlang. Unas características únicas de este paradigma esta diseñado para sistemas concurrentes, y distributivos, usa procesos ligeros para manejar la concurrencia y esta enfocado a la comunicación entre entidades y concurrentes.



Estándares de programación

Los estándares de programación son pautas usadas en la comunidad de desarrollo software que se acuerda seguir para escribir un código consistente y mantener la calidad y la legibilidad del software. Estos estándares ayudan a mejorar la legibilidad, la mantenibilidad y la colaboración entre programadores. Hay muchos estándares normalizados, pero estos son algunos de los principales y los más usados:

1. Estilo de codificación:

   Define la apariencia del código incluyendo el espaciado, el uso de mayúsculas y minúsculas, y la alineación. Algunos ejemplos en diferentes lenguajes de programación son PEP 8 en Python y Google java style para Java.

2. Nombrado de variables y Funciones:

   Define valores predeterminados para nombras diferentes variables haciendo más fácil la interpretación del código y facilitar la cooperación entre desarrolladores. Algunos nombres más comunes son camelCase o camel_case para nombres de variables y PascalCase para nombres de clase en diferentes lenguajes.

3. Comentarios y documentación:

   Se establecen reglas sobre la documentación del código incluyendo el formato de los comentarios y la frecuencia. Algunos ejemplos son Javadoc y docstrings para java y Python.

4. Manejo de errores y excepciones:

   Se establecen acciones y practicas comunes a la hora de manejar errores y excepciones de manera coherente, incluye como y cuando usar declaraciones como try-catch o bloques try-except.

5. Manejo de versiones:

   Define como se tienen q gestionar las versiones del software utiliza sistemas de control como Git y establece practicas para escribir mensajes claros y descriptivos.
6. Seguridad:

   Establece unas normas a la hora de escribir código seguro y resistente a ataques, Hace el uso de un manejo seguro para los datos sensibles, la

validación de entrada y la prevención a ataques teniendo en cuenta las vulnerabilidades conocidas.

7. Uso de bibliotecas y frameworks:

   Se usan unas normas para controlar el uso de bibliotecas externas y frameworks dentro de un proyecto. Estas normas abarcan acciones como importar, configurar y extender funcionalidades de bibliotecas externas.

8. Estructura de directorios y organización de proyectos.

   Establece una estructura predeterminada para organizar un proyecto y sea más fácil la organización de archivos entre colaboradores. Esta norma incluye cosas como donde deben ubicarse los archivos fuente, las pruebas, la documentación, ect.

9. Formato de archivos y codificación de caracteres:

   Define un formato para los archivos de un proyecto y la codificación de caracteres (UTF-8) para hacer de su uso en el proyecto.

10. Pruebas y cobertura de código:

    Establece unas acciones para escribir pruebas unitarias de integración y funcionales, estos incluyen estándares para medir y mejorar la cobertura del código.

11. Estándares de diseño:

    Define unas reglas y practicas para el diseño del código, como el (SRP) principio de responsabilidad única y el principio de inversión de dependencias (DIP).

Depende del lenguaje usado estos estándares pueden ser diferentes y sus usos variar, si eres un programador puedes no adherirte a estos estándares que no va a pasar nada, pero a la hora de programar en un proyecto en conjunto estos estándares van a ayudar a entender el código entre desarrolladores facilitando el trabajo y haciendo un código mas efectivo y más fácil de mantener. Pero en cambio si decides no seguir con estos estándares trabajando en conjunto, tus compañeros de proyecto pueden tener problemas a la hora de entender tu código, invirtiendo tiempo en comprenderlo y podría generar mas problemas a la hora de manejar código.

Conclusión

Hasta lo más básico puede afectar a cómo funciona tu código o incluso al mantenimiento de este en el tiempo, saber los lenguajes de programación a los que puedes optar es muy importante ya que dependiendo del trabajo que se quiere realizar existen algunos lenguajes que pueden realizar lo que solicitas de una manera mas efectiva que otros, así que la elección del lenguaje q se va a usar para el proyecto es importante para el optimo funcionamiento. El uso de paradigmas es importante a la hora de trabajar en el código ya que se siguen unas pautas las cuales ayudan a lidiar con el problema con efectividad y solucionar el problema de la mejor forma y lo más rápido posible. Los estándares de programación son muy importantes a la hora de trabajar en conjunto sobre un código, ya que gracias a estos es más fácil entender el código de otros compañeros y si facilitar el trabajo sobre este y su correcto mantenimiento.

Referencias:

Chat gpt

https://speckle-porpoise-32d.notion.site/Introducci-n-a-la-programaci-n-906cf521f30f41d9893cd2098b41cd87

https://conceptobasicodecomputacion.weebly.com/lenguaje-de-alto-medio-y-bajo-nivel.html

Framework y Librerias

Indice:

1: Qué es un framework

El framework es un término que proviene del inglés y significa «marco de trabajo» o «estructura». En el ámbito de la programación, un framework es un conjunto de herramientas y librerías que se utilizan para desarrollar aplicaciones más fácilmente y de manera más eficiente.

Un framework es un conjunto de reglas y convenciones que se usan para desarrollar software de manera más eficiente y rápida. Estos marcos de trabajo se emplean para ahorrar tiempo y esfuerzo en el desarrollo de aplicaciones, ya que proporcionan una estructura básica que se puede utilizar como punto de partida. Además, los frameworks también ofrecen soluciones a problemas comunes en el desarrollo de software, lo que significa que los desarrolladores pueden centrarse en las funcionalidades específicas de su aplicación en lugar de perder tiempo resolviendo problemas técnicos.

2: Qué es una librería

Una librería de programación es una colección de código desarrollado previamente que los programadores pueden utilizar para desarrollar software de manera más ágil. Estas colecciones de código reutilizable suelen resolver problemas o necesidades comunes de desarrollo.

Al usar librerías, el desarrollador no tiene que construir todo desde cero y pueden utilizar las funcionalidades de estas para construir su software. Sin ellas, el desarrollo de software sería realmente lento e ineficiente.

Los grandes proyectos de software pueden usar cientos o incluso miles de librerías externas para agilizar el proceso de codificación.

Un ejemplo del uso de una librería podría ser la autenticación de usuarios en una aplicación de comercio electrónico donde los usuarios deben iniciar sesión. Por lo general, la mejor manera de hacerlo de manera confiable es usar una librería de autenticación en lugar de escribir el código de autenticación.

3: Comparación de ventajas de frameworks vs librerías

Frameworks y librerías son elementos fundamentales en el desarrollo de software, pero cumplen roles diferentes en el proceso. Aquí hay una comparación de las ventajas de frameworks y librerías:

Frameworks:

1. Estructura y Organización:

- Ventaja: Los frameworks proporcionan una estructura organizativa para el código, lo que facilita la comprensión y mantenimiento del software. Definen una arquitectura que guía el desarrollo.

2. Productividad:

- Ventaja: Al ofrecer una estructura y un conjunto de herramientas predefinidas, los frameworks pueden acelerar el desarrollo. Muchas tareas repetitivas y comunes ya están implementadas, lo que reduce el tiempo necesario para crear aplicaciones.

3. Consistencia:

- Ventaja: Los frameworks establecen patrones y convenciones que contribuyen a la consistencia en el código. Esto facilita la colaboración entre desarrolladores y mejora la legibilidad del código.

4. Escalabilidad:

- Ventaja Los frameworks suelen ser escalables, lo que significa que pueden adaptarse a proyectos pequeños y crecer para satisfacer las necesidades de aplicaciones más grandes y complejas.

5. Integración:

- Ventaja: Muchos frameworks están diseñados para integrarse con otras tecnologías y herramientas, facilitando la interoperabilidad y la implementación de características adicionales.

Librerías:

1. Flexibilidad:

- Ventaja: Las librerías son más flexibles y modulares. Puedes elegir utilizar solo las partes necesarias de una librería, sin comprometerte con la estructura o arquitectura general del software.

2. Control Total:

- Ventaja: Al utilizar librerías, los desarrolladores tienen un control más directo sobre la lógica de su aplicación. Esto es útil cuando se necesita una solución más personalizada o específica.

3. Menor Sobrecarga:

- Ventaja: En comparación con los frameworks, las librerías tienden a tener menos sobrecarga. No imponen una estructura predefinida, lo que puede ser beneficioso en proyectos más pequeños o específicos.

4. Actualizaciones Independientes:

- Ventaja: Puedes actualizar las librerías de forma independiente sin afectar el resto del código. Esto facilita la gestión de dependencias y permite adoptar nuevas características sin actualizar todo el proyecto.

5. Aprendizaje Gradual:

- Ventaja: Es más fácil aprender y adoptar librerías de forma gradual, ya que puedes incorporarlas en partes específicas de tu aplicación según sea necesario.

En resumen, la elección entre frameworks y librerías dependerá de los requisitos específicos del proyecto, la preferencia del equipo de desarrollo y la naturaleza del problema a resolver. Los frameworks son ideales para proyectos grandes y complejos, mientras que las librerías ofrecen más flexibilidad para soluciones más específicas y controladas.

4: Qué framework consideras que sería óptimo para una tienda online y por qué.

El framework que mas optimo veo para una tienda online es Shopify, no es un framework tradicional, pero es una plataforma de comercio electrónico completa y fácil de usar, que te permite enfocarte en la gestión de productos y ventas. Para una tienda de pocas dimensiones como en mi caso sería la mía te permite enfocarte en las ventas y la gestión de productos ya que no tienes que lidiar con mucha cantidad de información.

5: Justificación de por qué has elegido no incorporar un framework a tu tienda.

A la hora de hacer mi página responsive decidí no implementar un framework ya que para mí desarrollo personal veía mejor hacerlo sin ninguna herramienta de automatización y así mejorar mi habilidad a la hora de programar y entender código.

Informe de Testing y Pruebas de Código

Requerimientos del informe:

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

- Conceptos Básicos:
  - Definición y diferencia entre testing y pruebas de código.
  - Objetivos y beneficios de realizar pruebas.

- Tipos de Pruebas:
  - Descripción detallada de diferentes tipos de pruebas, incluyendo, pero no limitado a: pruebas unitarias, de integración, de sistema, de aceptación, de carga, de estrés, etc.
  - Herramientas populares asociadas a cada tipo de prueba.

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

- Automatización de Pruebas:
  - Introducción a la automatización y sus ventajas.
  - Herramientas y frameworks populares para la automatización 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.

- Conclusión:
  - Reflexión sobre la importancia de las pruebas y el testing en garantizar la calidad y confiabilidad del software.

● **Introducción:**

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

Las pruebas de código y el testing son una parte fundamentales en el desarrollo de software, haciendo así mas fácil la creación de aplicaciones robustas y confiables. Hacer este testing puede ayudar a varios puntos a la hora de revisar el código:

1. Identificación de Errores y Defectos:

El proceso de testing ayuda a identificar errores y defectos en el código antes de que la aplicación antes de sacar la versión final de esta. Esto hace que la versión final este perfectamente para el uso del usuario.

2. Aseguramiento de la Calidad del Software:

Las pruebas son útiles para el aseguramiento de la calidad del software al garantizar que la aplicación cumpla con los requisitos especificados. Este proceso ayuda a evitar fallos y problemas en la funcionalidad, mejorando así la experiencia del usuario.

3. Reducción de Costos a Largo Plazo:

La capacidad de detectas un erros en una fase temprana del desarrollo ayudara a la reducción de costes ya que será más fácil la programación de la misma y el mantenimiento.

4. Mejora de la Mantenibilidad:

Las pruebas bien ejecutadas facilitan la comprensión del código y contribuyen a su modularidad. Esto hace que el software sea más mantenible, ya que los desarrolladores pueden realizar cambios o actualizaciones sin temor a introducir errores no deseados.

5. Validación de Requisitos:

Las pruebas ayudan a validar que el software desarrollado cumple con los requisitos establecidos. Esto asegura que la aplicación entregada al cliente sea funcional y se ajuste a las expectativas, evitando posibles malentendidos o discrepancias.

6. Aumento de la Confianza del Usuario:

Un software sometido a pruebas exhaustivas inspira confianza en los usuarios finales. La presencia de menos errores y una mayor estabilidad contribuye a una experiencia de usuario positiva, lo que puede ser crítico para el éxito y la adopción del software.

7. Facilita la Integración Continua:

Las pruebas son esenciales para la implementación de prácticas como la integración continua y la entrega continua (CI/CD), permitiendo una implementación más rápida y frecuente de nuevas funcionalidades sin comprometer la estabilidad del software.

8. Cumplimiento de Estándares y Normativas:

En algunos sectores, como la salud, finanzas o seguridad, es crucial cumplir con estándares y normativas específicas. Las pruebas ayudan a asegurar que el software cumpla con estos requisitos, garantizando la conformidad y la seguridad.

En resumen, las pruebas de código son esenciales para garantizar la calidad, fiabilidad y eficiencia del software, contribuyendo de manera significativa a todo el ciclo de vida del desarrollo y a la satisfacción del usuario final.

● Conceptos Básicos

■ Definición y diferencia entre testing y pruebas de código.

Definición:

- Testing: Es el proceso general de evaluación y verificación del software para identificar defectos y asegurar que cumple con los requisitos especificados.

- Pruebas de Código: Se refiere específicamente a la evaluación de la calidad del código fuente, buscando defectos, errores y asegurando buenas prácticas de programación.

Diferencias:

- testing: Incluye diversas actividades, como pruebas de unidad, integración, sistema y aceptación, abarcando diferentes niveles del desarrollo de software.

- Pruebas de Código: Se centra exclusivamente en la revisión y evaluación del código fuente, buscando mejorar su calidad, legibilidad y mantenibilidad.

■ Objetivos y beneficios de realizar pruebas.

Objetivos de Realizar Pruebas:

Los objetivos de realizar unas pruebas a nuestro software son la búsqueda de erres o fallos, la optimización del código ya que podemos observar márgenes de mejora donde se podría implementar algo y simplemente mejorar su rendimiento respecto a la versión final de presentación al usuario.

-Beneficios de Realizar Pruebas:

Sus principales beneficios son la resolución de problemas y errores a una fase temprana del desarrollo facilitando así su resolución y abaratando costes de empleados, también se busca facilitar su mantenimiento a largo plazo ya que será más fácil de entender el código y se encontrará con menos errores a lo largo de su existencia gracias a las pruebas realizadas.

● **Tipos de Pruebas:**

■ **Descripción detallada de diferentes 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.

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.

3. 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.

4. 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.

5. 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.

6.Prueba de carga

Se basa en enviar una gran cantidad de información buscando superar los limites del propio sistema buscando el fallo de la maquina para conocer sus limites y que cantidad de estrés puede soportar nuestro software, estas pruebas son muy comunes en servidores los cuales tienen que aguantar un gran trafico de usuarios e información al mismo tiempo sin poder permitirse un fallo.

■Herramientas populares asociadas a cada tipo de prueba.

-Pruebas unitarias

Junit: es un framework de código abierto desarrollado especialmente para crear, ejecutar y hacer reportes de estado de conjuntos de Prueba Unitaria automatizadas hechos en lenguaje Java.

Cactus: es un simple framwork de pruebas para Prueba Unitaria de código Java (Servlets, EJB, TagLib, etc). Cactus intenta simplificar la escritura de pruebas del lado del servidor. Usa JUnit y lo extiende.

EasyMock: proporciona Mock Object para interfaces durante la Prueba Unitaria mediante la generación de estas sobre la marcha utilizando mecanismo de Proxy de Java. Debido al estilo único de registro de expectativas en EasyMock, la mayoría de Refactor's no afectará a los objetos Mock. Así EasyMock es perfecto para Test Driven Development.

Mockito: es una librería Java para la creación de Mock Object muy usados para el testeo unitario en Test Driven Development, basado en EasyMock.Mockito fue creado con el objetivo de simplificar y solucionar algunos de los temas antes mencionados. EasyMock y Mockito puede hacer exactamente las mismas cosas, pero Mockito tiene un API más natural y práctico de usar.

MockEjb: es un framework de Software Libre liviano para ejecutar EJB. MockEjb implementa el API javax.ejb y crea implementaciones de Home y EJBObject para los EJB. Internamente, MockEjb utiliza proxy dinámicos e interceptores.

Spring Framework: contiene un conjunto de clases pensadas para realizar Prueba Unitaria y Prueba De Integracion, facilitando varias tareas repetitivas. En particular, estas utilidades se integran directamente con JUnit.

Jetty: es un proyecto de Software Libre que permite generar Mock Object de un servidor HTTP, un cliente HTTP y un contenedor de Servlets.

Dumbster: es un proyecto de Software Libre que permite generar Mock Object de un servidor de mail SMTP.Permite realizar Tests con Junit en los que se pueden enviar y obtener mails de este servidor de correo mock.

-Pruebas de intagracion

Selenium: entorno de pruebas de software para aplicaciones basadas en la web.

SoapUI: para la realización de pruebas a aplicaciones con arquitectura orientada a servicio (SOA) y transferencia de estado representacional (REST).

Postman: herramienta para hacer peticiones a APIs y generar colecciones de peticiones que nos permitan probarlas de una manera rápida y sencilla.

Watir: una familia de bibliotecas bajo la Licencia BSD para el lenguaje de programación Ruby que automatiza la operación de los navegadores web.

Mocha: framework de pruebas de JavaScript que se ejecuta en Node.js. Nos da la posibilidad de crear tanto tests síncronos como asíncronos de una forma muy sencilla.

Jasmine: framework para realizar pruebas unitarias y de integración de código JavaScript. Puede ser ejecutado en el navegador o sin él.

Jest: framework basado en Jasmine pero que posee muchas más funcionalidades.

Ava: framework como Mocha que permite correr nuestros tests de manera paralela en procesos aislados.

-Pruebas deaceptación

Canoo es una herramienta de Software Libre para automatizar las pruebas de aplicaciones web de una forma muy efectiva.

Concordion es un framework Java de Software Libre que permite convertir especificaciones en texto común sobre requerimientos en pruebas automatizadas.

FitNesse es una herramienta colaborativa para el desarrollo de software. Permite a los clientes, testers, y programadores aprender lo que su software debería hacer, y automáticamente compara lo que realmente hace. Compara las expectativas de los clientes a los resultados reales.

JBehave es un framework Java para mejorar la colaboración entre desarrolladores, QA, analistas de negocio, Dueño Del Producto y todos los miembros del equipo a través de escenarios automatizados y legibles.

JMeter es un proyecto de Apache Jakarta que puede ser usado para realizar una Prueba De Carga, para así analizar y mediar la Performance De Aplicaciones de distinto tipo, aunque con especial foco en Aplicaciones Web.

Sahi es una herramienta de automatización de pruebas para aplicaciones Web, con la facilidad de grabar y reproducir scripts. Está desarrollada en Java y JavaScript, la herramienta usa simplemente JavaScript para ejecutar los eventos en el browser.

Selenium es una herramienta de Software Libre para pruebas de aplicaciones Web. Las pruebas de Selenium se ejecutan directamente en un navegador y facilitan las pruebas de compatibilidad en navegadores, también como pruebas funcionales de aceptación de aplicaciones Web.

-Pruebas de rendimiento

Google Analytics

Es un servicio que captura información del tráfico sobre un sitio web y la actividad de los usuarios, como la duración de las sesiones, las páginas más visitadas y las acciones más frecuentes. Este servicio ya debe estar implementado en la aplicación para poder acceder a los datos. En base a esta información, se pueden elaborar pruebas más específicas y representativas de la demanda real de la aplicación y la actividad de los usuarios.

Http Logs Viewer

Permite filtrar y analizar de manera simple los archivos de logs de servidores Apache, IIS y Nginx. La herramienta brinda estadísticas con información sobre las consultas más ejecutadas, entre otras.

WebLog Expert

Permite procesar archivos de log y obtener estadísticas sobre la actividad de los usuarios en la aplicación, como por ejemplo, los archivos y pantallas que accedieron, las rutas que siguieron y browser que utilizaron, entre otros datos que se presentan gráficamente en el reporte generado.

Para utilizar estas herramientas es necesario solicitar los archivos de logs de acceso al administrador de la aplicación y saber a qué periodo de tiempo corresponden. Es importante entender que la información disponible puede variar y dependerá de la configuración de cada servidor.

-Pruebas de humo

Prueba de humo manual: Involucra la acción de una persona. A pesar de ser más lenta, tiene la ventaja de observar la interacción de un usuario con la aplicación o plataforma.

Prueba de humo automatizada: Se utilizan herramientas de automatización para el testing de software. Con este método, se ahorra tiempo y se gestionan las tareas de manera más eficiente.

Prueba de humo híbrida: Se combina las mejores prácticas de la prueba manual y la prueba automatizada. Con esta metodología, se obtiene la rapidez de la automatización y la calidad de la valoración de la interacción del usuario con el software, lo que la convierte en la mejor opción para realizar una prueba de humo.

-Prueba de carga

LoadView

Es una herramienta de prueba de carga fácil de configurar y rica en funciones que supera a todos los demás competidores de la industria. Una sólida experiencia del cliente, precisión de localización y excelente atención al cliente distinguen a esta herramienta. Sus mejores características incluyen poder probar en cualquier nivel. También elimina la necesidad de infraestructura de backend, que puede ser difícil de configurar. Una empresa centrada en el usuario, proporciona a los usuarios un gran panel de control que es lógico e intuitivo. Además, los resultados que proporciona son fáciles de entender y se utilizan rápidamente.

Apache JMeter

Esta herramienta mide la funcionalidad y el rendimiento de los sitios web y varias otras aplicaciones. Es totalmente de código abierto y se ejecuta en el script Java. Es un buen recurso para probar los niveles de rendimiento y el comportamiento de los sitios web. Hay diferentes tipos de carga disponibles, lo que da una idea de cuánto estrés puede soportar un sitio web.

LoadNinja

Se jacta de poder reducir el tiempo de prueba a la mitad, ya que no se necesitan scripts de prueba. El usuario puede centrarse simplemente en crear pruebas de carga. Con esta herramienta, el usuario puede medir el rendimiento de su sitio web con métricas y pasos de acción disponibles en cualquier navegador. Esta herramienta puede ayudar a identificar problemas de rendimiento y puede proporcionar detalles sobre la experiencia del usuario final. Las pruebas son escalables en cualquier grado.

Radview WebLOAD

Esta es una herramienta de prueba de carga y rendimiento de nivel empresarial. Su capacidad para realizar grandes pruebas de carga lo distingue de la competencia. Permite un número ilimitado de usuarios simulados y luego produce un informe para las áreas débiles que deben abordarse. Esta herramienta de prueba de carga es ideal para cualquier gran empresa o corporación que tenga una alta carga de usuarios.

BlazeMeter

Una plataforma de pruebas continuas que permite probar aplicaciones web. BlazeMeter puede funcionar con herramientas de prueba de código abierto. Debido a que es fácil de usar, los usuarios menos experimentados pueden comenzar a probar el rendimiento de sus sitios web en esta herramienta de inmediato. Esta herramienta también permite colaboraciones.

■ Exploración de técnicas como TDD (Test Driven Development), BDD (Behavior Driven Development) y otras relevantes.

■ Beneficios y retos asociados a cada técnica.

TDD (Test Driven Development):

Descripción:

En TDD, los tests unitarios se escriben antes de la implementación del código funcional. El ciclo se compone de tres pasos: escribir un test que falle, implementar el código mínimo para que el test pase y luego refactorizar el código.

Beneficios:

1. Detección temprana de errores: Los errores se identifican en las etapas iniciales del desarrollo, facilitando su corrección.

2. Documentación viva: Los tests sirven como documentación actualizada y ejemplos de uso del código.

3. Diseño modular: Fomenta la creación de código modular y fácilmente mantenible.

4. Confianza en los cambios Facilita la introducción de cambios al proporcionar una red de seguridad a través de tests.

Retos:

1. Curva de aprendizaje inicial: Puede ser desafiante para aquellos nuevos en la metodología.

2. Necesidad de reestructuración:** Puede requerir modificaciones significativas en la forma de abordar el diseño y desarrollo del software.

BDD (Behavior Driven Development):

Descripción:

BDD se centra en la colaboración entre desarrolladores, testers y no técnicos para describir y comprender el comportamiento esperado del sistema. Utiliza un lenguaje más cercano al negocio y se apoya en herramientas como Cucumber o SpecFlow.

Beneficios:

1. Colaboración mejorada: Involucra a stakeholders no técnicos en el proceso de especificación y validación del comportamiento.

2. Lenguaje común: Utiliza un lenguaje que puede ser entendido tanto por técnicos como no técnicos, mejorando la comunicación.

3. Enfoque en el comportamiento: Centrado en el comportamiento del sistema, asegurando que la funcionalidad se alinee con las expectativas del usuario final.

Retos:

1. Necesidad de habilidades de comunicación: Requiere habilidades para traducir requisitos en escenarios de comportamiento comprensibles.

2. Costo inicial de implementación: La creación de lenguaje de negocio específico y la configuración inicial pueden llevar tiempo y recursos.

Otras Técnicas Relevantes:

1. Pruebas de Integración Continua (CI):

- Beneficios: Garantiza que las diferentes partes del sistema trabajen correctamente juntas.

- Retos: Puede ser complicado mantener un entorno de integración continua robusto y rápido.

2.Pruebas de Aceptación Automatizadas:

- Beneficios: Automatiza la validación de que el software cumple con los criterios de aceptación del usuario.

- Retos Puede ser complejo mantener y actualizar scripts de pruebas automatizadas.

3. Pruebas Exploratorias:

- Beneficios: Descubre problemas no contemplados en los casos de prueba planificados.

- Retos: Puede no ser tan estructurado y puede depender en gran medida de la habilidad y experiencia del tester.

4. Pruebas de Estrés y Rendimiento:

- Beneficios: Identifica cómo se comporta la aplicación bajo condiciones extremas.

- Retos: Puede ser difícil simular con precisión situaciones de carga real.

- **Automatización de Pruebas:**
  - **Introducción a la automatización y sus ventajas.**

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. Todo esto nos lleva a una serie de ventajas:

-Ahorro de tiempo y esfuerzo. Por ejemplo, en un sprint se pueden incluir varias funcionalidades nuevas que deben ser probadas, pero además de esto, se debe verificar que éstas no hayan afectado el sistema en general. En estas ocasiones, ejecutar una regresión es siempre muy útil. Si los casos de pruebas están automatizados, éstos se ejecutan y los testers solamente deben revisar los resultados, lo cual les permite enfocarse en las pruebas manuales de las nuevas funcionalidades.

-Verificar que los errores pasados no se reproduzcan. En caso que un error haya sido corregido, automatizar el caso de prueba que lo verifica puede asegurar que este error no sea introducido nuevamente por parte de algún cambio en otra funcionalidad.

-Verificar que el producto sea funcional para el usuario. Al enfocar los casos de prueba en la funcionalidad mínima del software, el equipo de QA puede asegurar que la nueva versión no vaya a afectar el mínimo útil para el usuario. Automatizar estos casos de prueba es de gran ayuda ya que al estar en etapas cercanas al lanzamiento, los testers se pueden concentrar en la nueva funcionalidad mientras que las pruebas de aceptación se ejecutan de forma automatizada.

-Verificar que no haya ninguna funcionalidad rota en el ‘build’. Al desplegar una versión del producto, ya sea una nueva funcionalidad o alguna corrección, se debe verificar el ‘build’. En este proceso de integración continua/despliegue continuo (CI/CD por sus siglas en inglés), la automatización de casos de prueba enfocados en la verificación del ‘build’ es de gran ayuda ya que garantizan una funcionalidad mínima.

  - **Herramientas y frameworks populares para la automatización de pruebas.**

- **Casos de Uso y Ejemplos:**

SERENITY(JAVA)

Es un framework de pruebas automatizadas basado en Java que se integra con herramientas behavior-driven development (BDD) como Cucumber y JBehave permitiendo tener los escenarios como información de alto nivel mientras también provee información detallada de los mismos en los reportes. Esta pensado principalmente para pruebas de aceptación y regresión.

CYPRESS(JAVASCRIPT)

Este framework se centra mas que nada en los desarrolladores, facilitando la practica de TDD(test-driven development)

ROBOT FRAMEWORK(PYTHON)

Es el framework numero 1 para pruebas automatizadas en Python. Es un Framework del tipo keyword-driven, lo que hace que los tests sean fáciles de leer y crear. Si no usas Python, podes utilizarlo igual gracias a Jython (Java) o IronPython (.NET).

GALEN FRAMEWORK (JAVASCRIPT/JAVA)

Este framework esta pensado mayormente en las pruebas de interfaz de usuario. Tiene una sintaxis especial y reglas que pueden utilizarse para describir y verificar la interfaz de usuario de la aplicación. Tiene la posibilidad de generar reportes HTML con screenshots y herramientas de comparación de imágenes.

GAUGE(JAVA/RUBY/C#)

Enfocado en BDD, permite la creación de documentación basándonos en las pruebas que se ejecutan. Permite utilizar plugins para extender sus funcionalidades.

CITRUS FRAMEWORK(JAVA)

Este framework te permite realizar pruebas automatizadas de integración de cualquier protocolo de comunicación o formato de datos. Esto hace mas simple las pruebas de escenarios donde la aplicación interactua mas que nada con servicios de terceros.

Si se requiere pruebas de Interfaz de Usuario, se integra fácilmente con Selenium. Esto permite realizar pruebas end-to-end completas incluyendo las interacciones en el Backend de la aplicación.

KARATE-DSL(JAVA)

Pensado principalmente para pruebas de servicios, esta herramienta implementa BDD para las pruebas API de tu aplicación. Se integra fácilmente con Cucumber y permite extender nuestras pruebas BDD de la UI a servicios.

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

1. TDD en Desarrollo de Software:

- Contexto: Desarrollo de una nueva funcionalidad en un sistema de gestión de ventas en línea.

- Aplicación: Se escriben pruebas unitarias antes de implementar la lógica de la funcionalidad. Las pruebas aseguran que la funcionalidad cumple con los requisitos y que futuros cambios no introducirán regresiones.

2. BDD en Desarrollo de una Aplicación Móvil:

- Contexto: Desarrollo de una aplicación móvil de seguimiento de ejercicio.

- Aplicación: Se utiliza BDD para definir comportamientos esperados, como el registro de actividades y la generación de informes. Stakeholders no técnicos colaboran en la definición de escenarios de comportamiento.

3. Pruebas de Integración Continua (CI) en un Proyecto Web:

- Contexto: Desarrollo de un sitio web de comercio electrónico.

- Aplicación: Se implementa CI para verificar la integración de componentes críticos como carrito de compras, procesamiento de pagos y gestión de inventario. Se ejecutan pruebas automáticas para asegurar que las actualizaciones no afecten negativamente la funcionalidad.

4. Pruebas de Aceptación Automatizadas en un Sistema de Gestión de Recursos Humanos:

- Contexto: Desarrollo de un sistema de gestión de recursos humanos.

- Aplicación: Se crean pruebas automatizadas para verificar el flujo de aprobación de solicitudes de vacaciones, asegurando que el sistema cumple con los criterios de aceptación definidos por los usuarios finales.

5. Pruebas Exploratorias en una Aplicación de Red Social:

- Contexto: Desarrollo de una aplicación de red social.

- Aplicación: Se realizan pruebas exploratorias para descubrir posibles problemas de usabilidad, como la dificultad de navegación o la falta de claridad en la interfaz de usuario. Los testers exploran la aplicación de manera libre para identificar posibles mejoras.

6. Pruebas de Estrés y Rendimiento en una Plataforma de Transacciones Financieras:

- Contexto: Desarrollo de una plataforma de transacciones financieras en línea.

- Aplicación: Se llevan a cabo pruebas de estrés para evaluar cómo la plataforma maneja un gran volumen de transacciones simultáneas. Se miden los tiempos de respuesta y la estabilidad del sistema bajo cargas extremas.

Estos ejemplos ilustran cómo diversas técnicas y tipos de pruebas se aplican en contextos específicos para garantizar la calidad y fiabilidad del software en proyectos del mundo real. La selección de la técnica de prueba apropiada depende del objetivo específico y las características del proyecto.

● **Conclusión:**

■ **Reflexión sobre la importancia de las pruebas y el testing en garantizar la calidad y confiabilidad del software.**

El testing de software es una de las actividades más importantes y fundamentales en el desarrollo de un proyecto, ya que posibilita los procesos, métodos de trabajo y herramientas necesarias para garantizar la calidad de cualquier desarrollo. Sin embargo, hoy en día continuamos encontrándonos con problemas a la hora de comprender el testing de software como una inversión, incluso en los proyectos más tradicionales donde asumíamos que las actividades de pruebas estaban normalizadas.

Con el fin de poder detectar a tiempo cualquier error y garantizar que el producto cumple con todas las premisas establecidas, el testing debe existir en todas las fases de un proyecto: desde la toma de requerimientos en cliente, pasando por las reuniones de seguimiento, hasta la entrega del producto final. Es más, un proyecto carente de este proceso en todas sus fases acaba generando un mayor coste económico y un mayor esfuerzo durante la fase de pruebas.

●**Referencias**

Para citar las fuentes en formato Harvard, puedes seguir el siguiente formato para cada enlace:

1. Pablos, J. (2018, mayo 2). Top 7 Frameworks de Pruebas Automatizadas. [Enlace](https://josepablosarco.wordpress.com/2018/05/02/top-7-frameworks-de-pruebas-automatizadas/).

2. Encora. (s.f.). Testing de Software: La Importancia de Automatizar los Casos de Prueba. [Enlace](https://www.encora.com/es/blog/testing-de-software-la-importancia-de-automatizar-los-casos-de-prueba).

3. Abstracta. (s.f.). Automatizar Pruebas de Software. [Enlace](https://cl.abstracta.us/blog/automatizar-pruebas-de-software/#:~:text=La%20automatizaci%C3%B3n%20de%20las%20pruebas,nos%20ocupamos%20de%20otras%20actividades.).

4. Atlassian. (s.f.). Automated Testing - Software Testing. [Enlace](https://www.atlassian.com/es/continuous-delivery/software-testing/automated-testing#:~:text=Las%20pruebas%20automatizadas%20consisten%20en,lleva%20a%20cabo%20una%20persona.).

5. Zaptest. (s.f.). Smoke Testing: Profundización en Tipos, Proceso, Herramientas de Software de Smoke Test y Más. [Enlace](https://www.zaptest.com/es/smoke-testing-profundizacion-en-tipos-proceso-herramientas-de-software-de-smoke-test-y-mas).

6. Qalified. (s.f.). Descubre las Mejores Herramientas para tus Pruebas de Rendimiento. [Enlace](https://qalified.com/es/blog/descubre-las-mejores-herramientas-para-tus-pruebas-de-rendimiento/#:~:text=Gatling%20y%20Taurus.-,JMeter,mundo%20para%20pruebas%20de%20rendimiento.).

7. Sogeti. (s.f.). Pruebas de Aceptación de Usuario. [Enlace](https://www.sogeti.es/soluciones/calidad-de-software/servicios-de-testing/testing/pruebas-de-aceptacion-de-usuario/).

8. UnaQAenApuros. (2021, abril 28). 072 Pruebas de Integración III - Herramientas. [Enlace](https://unaqaenapuros.wordpress.com/2021/04/28/072-pruebas-de-integracion-iii-herramientas/).

9. Geekflare. (s.f.). 32 Herramientas Populares de Pruebas de Software para 2021. [Enlace](https://geekflare.com/es/software-testing-tools/).

10. Atlassian. (s.f.). Types of Software Testing. [Enlace](https://www.atlassian.com/es/continuous-delivery/software-testing/types-of-software-testing).

Recuerda ajustar la fecha de acceso para cada enlace si la información es dinámica y puede cambiar con el tiempo.