# Sesión 4 : Prácticas de Seguridad Recomendadas para el Desarrollo de Software

Introducción a la seguridad de aplicaciones y mejores prácticas en codificación #

# Mauricio Sotelo

- Ingeniero de Seguridad Nivel III
- Seguidor de las buenas prácticas, marcos de trabajo y estándares de seguridad.
- Amo las actividades al aire libre y hacer todo con música de fondo.
- https://calendly.com/mauricio_sotelo

# Recomendaciones importantes

Identifícate en Zoom usando tu nombre y apellido.

Silencia tu micrófono durante el curso.

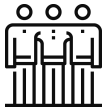
Utiliza el chat para hacer preguntas en la sección asignada para ello.

Centra tus preguntas en el tema presentado.

Mantén tu cámara encendida durante toda la sesión.

# Academy Codigo de Conducta

Sé respetuoso, no hay preguntas o ideas malas.



Sé empático y paciente.

Sé cuidadoso con las palabras que eliges.

# Objetivo

**Al final de esta sesión podrás:**

- Comprender la importancia de la seguridad en el desarrollo de software.
- Aplicar principios fundamentales de codificación segura, como revisiones de código colaborativas, pruebas unitarias y auditorías de seguridad, para identificar, rectificar y prevenir vulnerabilidades y errores.

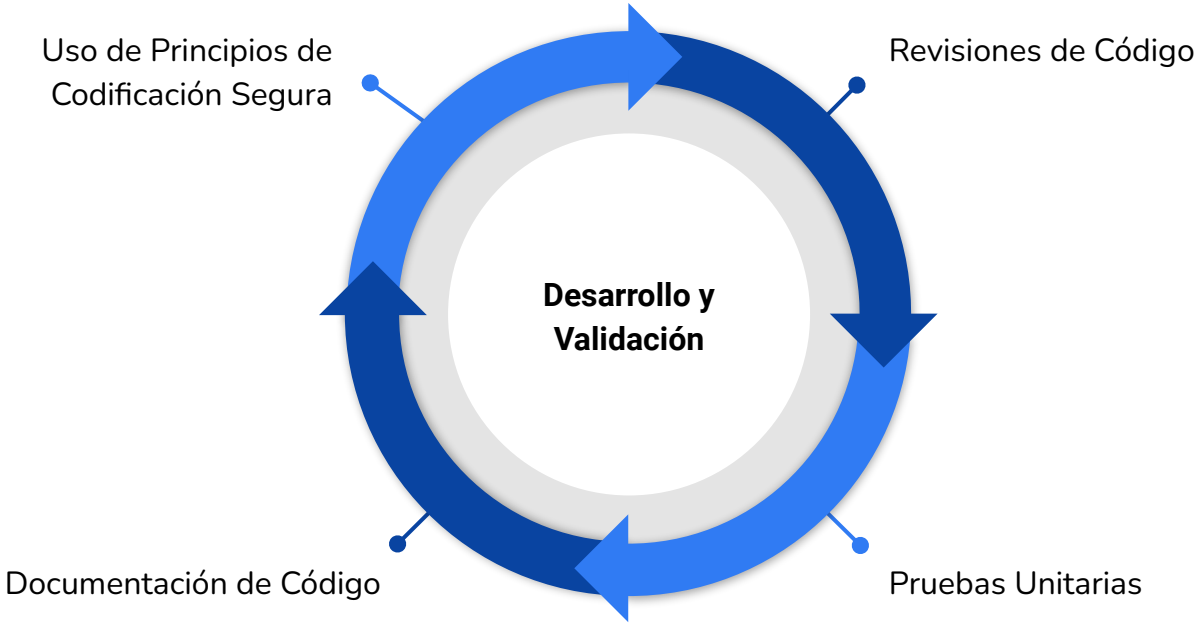
# Tabla de Contenidos

# Prerrequisitos

# Prerrequisitos

1. Entorno de Control de Versiones.
2. Convenciones de Código.
3. Formación en Seguridad.
4. Herramientas de Revisión Automatizada.
5. Proceso Definido.
6. Cultura de Feedback Constructivo.
7. Tiempo y Recursos.
8. Compromiso del Equipo y la Gerencia.
9. Formación Continua.
10. Herramientas de Comunicación.

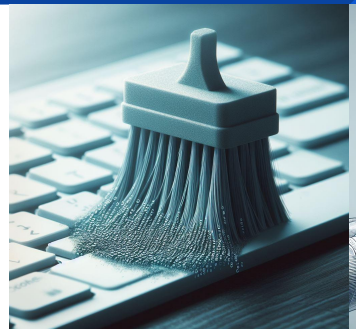
# Prácticas de Seguridad para el Desarrollo de Software



# Revisiones de Código

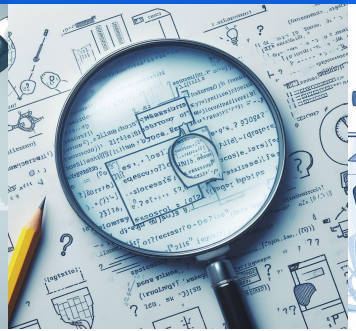


Preparación

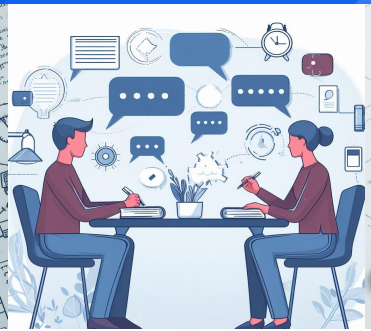


Solicitud de Revisión

Revisión



Discusión



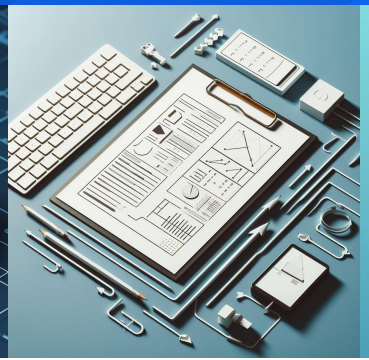
Aprobación y Fusión

# Pruebas Unitarias

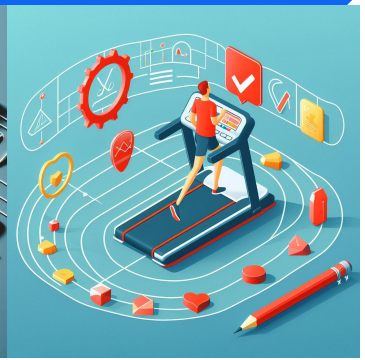
Identificar Componentes a Probar

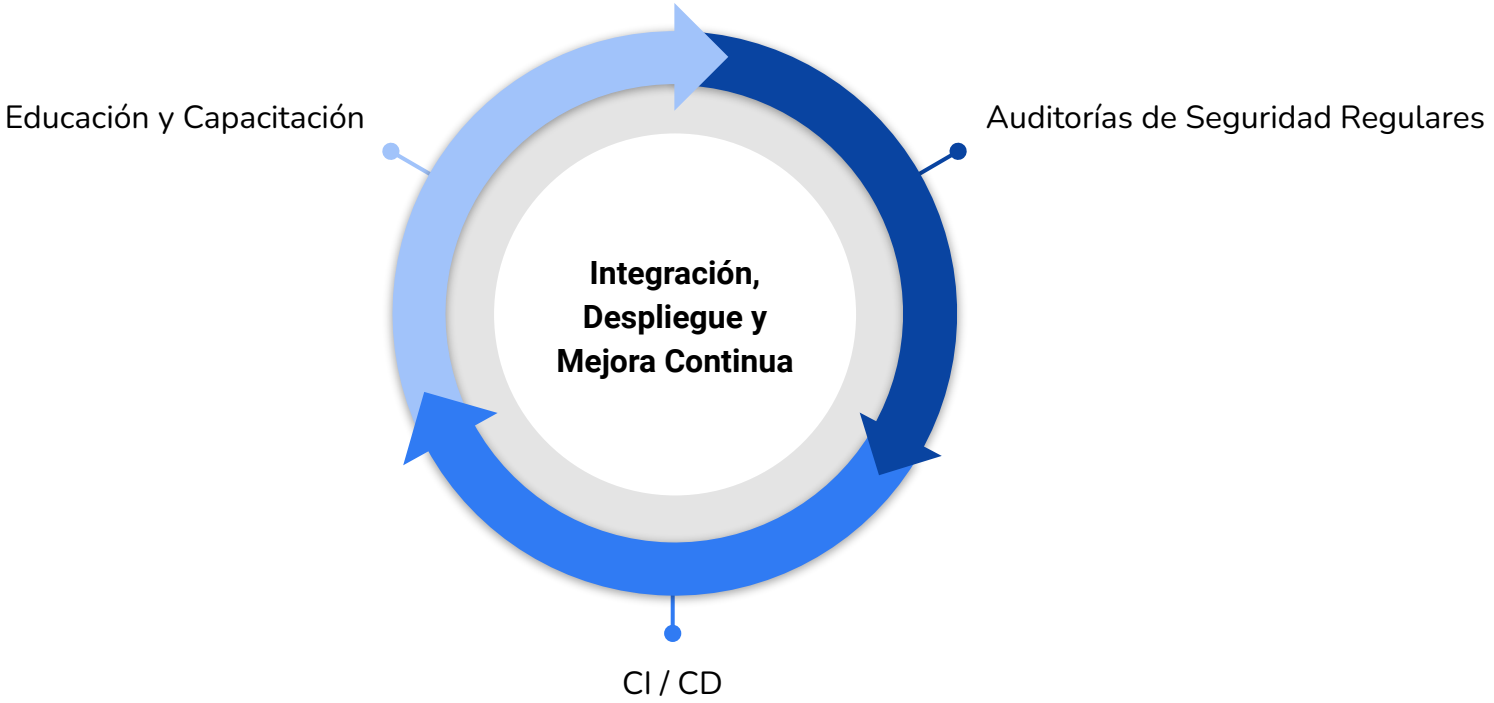
Crear el Entorno de Pruebas

Escribir Pruebas



Ejecutar y Revisar

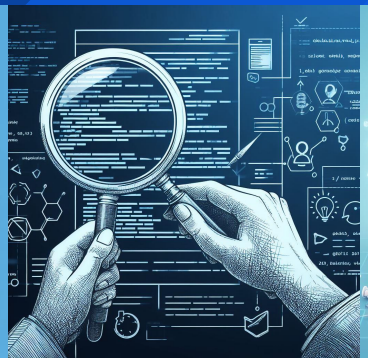




# Auditorías de Seguridad Regulares

Usar Herramientas Automatizadas

Inspecciones Manuales

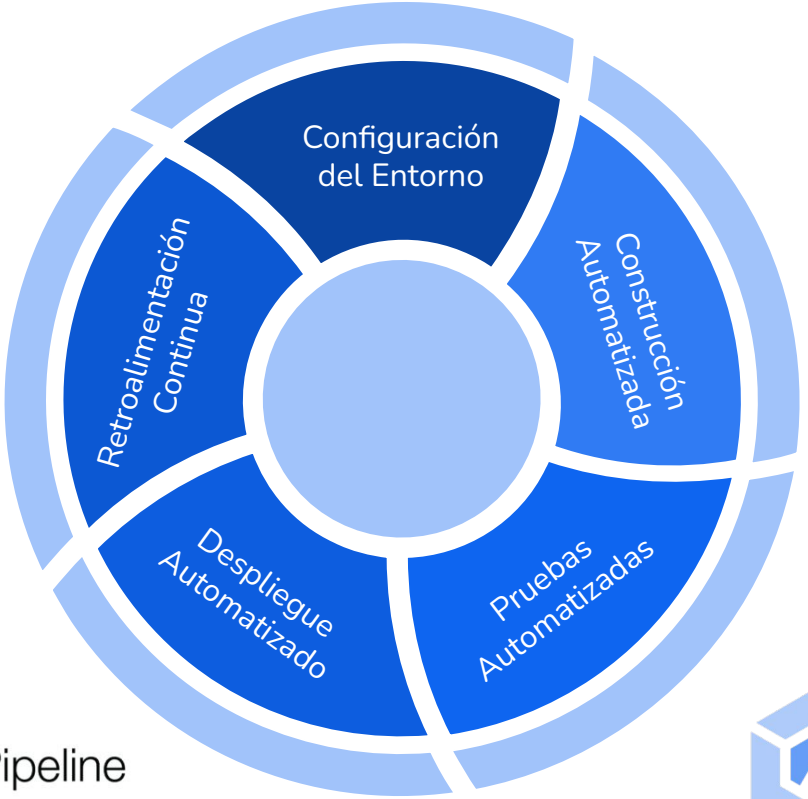


Actualizar y Parchar

Registros de Auditoría

# Integración Continua y Despliegue Continuo (CI/CD)

Azure DevOps




AWS CodePipeline

Google Cloud Build

# Descanso

5 minutos.

# Desafios

```
namespace WebApp
{
    public class ImageUpload
    {
        // Función para cargar imágenes
        public void UploadImage(string filePath)
        {
            // Supongamos que este método se encarga de cargar la imagen desde un path
            byte[] imageBytes = File.ReadAllBytes(filePath);
      // VALIDACIÓN INCOMPLETA: Solo verifica la extensión del archivo
            if (Path.GetExtension(filePath) == ".jpg")
            {
                SaveImageToDatabase(imageBytes);
            }
            else
            {
                Console.WriteLine("Ti
            }
        }
        private void SaveImageToDatab
        {
            // Lógica para guardar la
            Console.WriteLine("Imagen
        }
    }
}
```

→ **Problema**: La función UploadImage verifica únicamente la extensión del archivo para decidir si es una imagen. Un atacante podría fácilmente renombrar un archivo malicioso con extensión .jpg y pasarlo como si fuera una imagen legítima.

→ **Revisión de código**: Un revisor señalaría que verificar únicamente la extensión no es una validación segura.

→ **Revisión de código**: Una validación más robusta analizaría el contenido del archivo para asegurarse de que es una imagen legítima y no contiene ningún código malicioso.

→ **Corrección sugerida**: Utilizar una biblioteca especializada en procesamiento de imágenes para leer y validar que el contenido del archivo es realmente una imagen.

# ¿ Cómo lo corregirías ?

```
using System;
using System.Drawing;
using System.IO;

namespace WebApp
{
    public class ImageUpload
    {
        // Función para cargar imágenes
        public void UploadImage(string filePath)
        {
            byte[] imageBytes = File.ReadAllBytes(filePath);

            if (IsValidImage(filePath))
            {
                SaveImageToDatabase(imageBytes);
            }
            else
            {
                Console.WriteLine("Tipo de archivo no permitido o no es una imagen válida.");
            }
        }
```

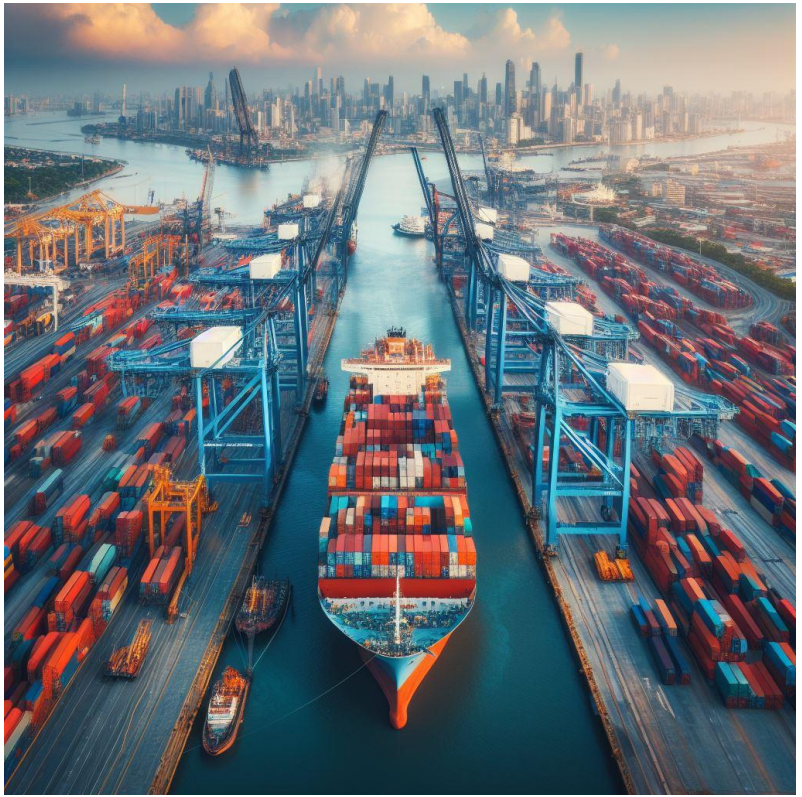
```
        // Función para validar si el archivo es realmente una imagen
        private bool IsValidImage(string filePath)
        {
            try
            {
                using (Image image = Image.FromFile(filePath))
                {
                    // Si podemos cargar la imagen sin errores, es una imagen válida
                    return true;
                }
            }
            catch
            {
                // Si hay un error al cargar el archivo como imagen, no es una imagen válida
                return false;
            }
        }

        private void SaveImageToDatabase(byte[] imageBytes)
        {
            // Lógica para guardar la imagen en la base de datos
            Console.WriteLine("Imagen guardada correctamente.");
        }
    }
}
```

## Prueba Estática: Sistema de Gestión de Itinerarios Marítimos.

A. **Contexto**: La aplicación permite a los operadores del puerto añadir y consultar los itinerarios de barcos que llegan y salen.

B. **Instrucciones**: En el siguiente fragmento de código, se han detectado cinco errores que pudieron ser escritos de manera deliberada por un programador que está en contra de las prácticas de seguridad en el desarrollo de software. Identifícalos y propón la solución para cada uno de dichos errores.



```
using System;
using System.Data.SqlClient;

public class MaritimeScheduleSystem
{
    private string _connectionString = "Data Source=localhost;Initial
Catalog=PortSchedules;User Id=admin;Password=12345;";

    public void AddShipSchedule(DateTime arrivalDate, string shipName)
    {
        using (SqlConnection connection = new SqlConnection(_connectionString))
        {
            connection.Open();
            string query = $"INSERT INTO ShipSchedules(ArrivalDate, ShipName)
VALUES('{arrivalDate}', '{shipName}')";

            SqlCommand command = new SqlCommand(query, connection);
            command.ExecuteNonQuery();
        }
    }
```

```
public void DisplayAllSchedules()
    {
        try
        {
            using (SqlConnection connection = new SqlConnection(_connectionString))
            {
                connection.Open();
                string query = "SELECT * FROM ShipSchedules";
                SqlCommand command = new SqlCommand(query, connection);
                SqlDataReader reader = command.ExecuteReader();
                while (reader.Read())
                {
                    Console.WriteLine($"Fecha de Llegada: {reader["ArrivalDate"]}, Nombre del
Barco: {reader["ShipName"]}");
                }
            }
        }
        catch
        {
            Console.WriteLine("EPA Algo salió mal.");
        }
    }
}
```

## Errores Deliberados:

1. **Conexión Insegura a la Base de Datos**: El string de conexión muestra el nombre de usuario y la contraseña en texto plano.
2. **Inyección SQL**: El método AddShipSchedule utiliza concatenación de strings para construir la consulta SQL, lo que lo hace vulnerable a ataques de inyección SQL.
3. **Manejo Genérico de Errores**: El método DisplayAllSchedules tiene un bloque try-catch que solo muestra un mensaje genérico, sin dar información detallada sobre el error ni registrar el error en ningún lugar.
4. **Ausencia de Comentarios**: No hay comentarios en el código que expliquen qué hace cada método o cómo se deberían usar.
5. **Credenciales Fáciles de Adivinar**: Las credenciales utilizadas para la conexión a la base de datos ("admin" y "12345") son muy básicas y fáciles de adivinar. Susceptible a un ataque de Fuerza Bruta.

## Soluciones:

1. Utilizar credenciales cifradas y/o sistemas de autenticación más seguros.
2. Utilizar parámetros en las consultas SQL en lugar de concatenar strings.
3. Proporcionar un manejo de errores más detallado y, si es posible, registrar errores en un log sin exponer información sensible al usuario.
4. Añadir comentarios descriptivos en los métodos y puntos clave del código.
5. Cambiar las credenciales por unas más seguras y robustas.

# ¿Preguntas?

# Gracias.