Practica de construcción de software

TEMA: flujo de trabajo para el uso de Git

1. Descripción

Desarrollar una aplicación de gestión de tareas en Python utilizando el enfoque de Desarrollo Guiado por Pruebas (TDD) y aplicando el flujo de trabajo GitFlow. La aplicación debe incluir una interfaz gráfica (GUI) y permitir al usuario gestionar tareas simples. Debe incluir las siguientes funcionalidades:

1. Agregar una Tarea: Permitir al usuario agregar una nueva tarea con un título y una descripción.
2. Ver Tareas: Mostrar una lista de todas las tareas agregadas.
3. Marcar Tarea como Completada: Permitir al usuario marcar una tarea como completada.
4. Eliminar una Tarea: Permitir al usuario eliminar una tarea de la lista.

2. Instrucciones:

1. Utiliza el flujo de trabajo GitFlow para organizar tu código en diferentes ramas según las etapas del desarrollo: develop, feature, release, hotfix, etc.
2. Utiliza TDD para desarrollar cada funcionalidad de la aplicación. Escribe primero las pruebas unitarias para cada función que implementes y luego escribe el código para hacer pasar esas pruebas.
3. Implementa una interfaz gráfica utilizando Tkinter o PyQT (elige una).
4. Maneja los casos de entrada inválidos de manera adecuada, por ejemplo, títulos vacíos para las tareas.
5. Realiza commits frecuentes en tu repositorio Git, siguiendo el flujo de trabajo GitFlow.
6. Al finalizar el desarrollo, crea una versión de lanzamiento (release) y fusiona la rama de lanzamiento en master y develop.
7. tu código de manera clara y concisa, incluyendo comentarios y explicaciones sobre el diseño y funcionamiento de la aplicación.

3. Recursos

1. Puedes usar cualquier framework de pruebas unitarias de Python, como unittest o pytest, para implementar las pruebas.
2. Consulta la documentación oficial de GitFlow para comprender mejor su flujo de trabajo: GitFlow
3. Utiliza Git y GitHub para gestionar tu código y colaborar con otros en el desarrollo del proyecto.
4. Para la interfaz gráfica, puedes consultar la documentación de Tkinter o PyQt5.

4. Flujo de trabajo GitFlow

Este proyecto sigue el flujo de trabajo GitFlow para la gestión de ramas y versiones. Las ramas principales son:

- develop
- feature/*
- release/*
- hotfix/*
- master

5. Consulta la documentación de GitFlow para más detalles

```
### Proceso de Desarrollo:
1. **Inicializar el Repositorio Git y Crear Ramas**:
   ```bash
   git init
   git checkout -b develop
   git checkout -b feature/tareas```
```

1. Implementar las Funcionalidades Usando TDD:

   - Agregar las pruebas unitarias en tests/test_gestor_tareas.py.
   - Implementar las funcionalidades en gestor_tareas.py.
   - Realizar commits frecuentes.

2. Fusionar y Preparar el Lanzamiento:

```
git checkout develop
git merge feature/tareas
git checkout -b release/1.0.0
# Realizar pruebas adicionales y ajustes
git checkout master
git merge release/1.0.0
git tag -a v1.0.0 -m "Lanzamiento de la versión 1.0.0"
git checkout develop
git merge release/1.0.0
```

3. Corregir Errores Críticos si es Necesario:

```
git checkout -b hotfix/1.0.1
# Implementar correcciones
git checkout master
git merge hotfix/1.0.1
git tag -a v1.0.1 -m "Lanzamiento de la versión 1.0.1"
git checkout develop
git merge hotfix/1.0.1
```

4. Generar el archivo requirements.txt

```
# Crear un entorno virtual
python -m venv env

# Activar el entorno virtual
source env/bin/activate  # En Windows usa `env\Scripts\activate`

# Instalar dependencias
pip install pyqt5

# Generar el archivo requirements.txt
pip freeze > requirements.txt
```
```

Solución Propuesta

Estructura de Archivos del Proyecto:

```
gestor_tareas/
├── tests/
│   └── test_gestor_tareas.py
├── src/
│   ├── logica
│   │   └── test_gestor_tareas.py
│   └── vista
│       └── gui_gestor_tareas.py
└── README.md
```

README.md:

```
# Gestor de Tareas

Esta es una aplicación de gestión de tareas desarrollada en Python. Permite agregar, ver, marcar
como completadas y eliminar tareas. La aplicación utiliza Tkinter para la interfaz gráfica y sigue
el enfoque de Desarrollo Guiado por Pruebas (TDD) junto con GitFlow para la gestión del código.

## Requisitos

- Python 3.x
- Tkinter (incluido en la instalación estándar de Python)

## Instalación

1. Clona este repositorio:
   ```bash
   git clone <URL_del_repositorio>```
```

Código Python (gestor_tareas.py)

```
import tkinter as tk
from tkinter import ttk, messagebox

class Tarea:
    def __init__(self, titulo, descripcion):
        self.titulo = titulo
        self.descripcion = descripcion
        self.completada = False

class GestorTareas:
    def __init__(self):
        self.tareas = []

    def agregar_tarea(self, titulo, descripcion):
        if not titulo:
            raise ValueError("El título no puede estar vacío")
        tarea = Tarea(titulo, descripcion)
        self.tareas.append(tarea)

    def obtener_tareas(self):
        return self.tareas

    def marcar_completada(self, indice):
        if 0 <= indice < len(self.tareas):
            self.tareas[indice].completada = True
        else:
            raise IndexError("Índice fuera de rango")

    def eliminar_tarea(self, indice):
        if 0 <= indice < len(self.tareas):
            del self.tareas[indice]
        else:
            raise IndexError("Índice fuera de rango")

class GestorTareasGUI:
    def __init__(self, root, gestor):
        self.gestor = gestor
        self.root = root
```
```

```
        self.root.title("Gestor de Tareas")

        self.frame = ttk.Frame(root, padding="10")
        self.frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        self.titulo_entry = ttk.Entry(self.frame, width=20)
        self.titulo_entry.grid(row=0, column=1, sticky=tk.W)

        self.descripcion_entry = ttk.Entry(self.frame, width=50)
        self.descripcion_entry.grid(row=1, column=1, sticky=tk.W)

        self.agregar_btn = ttk.Button(self.frame, text="Agregar Tarea", command=self.agregar_tarea)
        self.agregar_btn.grid(row=2, column=1, sticky=tk.W)

        self.tareas_listbox = tk.Listbox(self.frame, height=10, width=50)
        self.tareas_listbox.grid(row=3, column=1, sticky=tk.W)

        self.completar_btn = ttk.Button(self.frame, text="Marcar como Completada",
command=self.marcar_completada)
        self.completar_btn.grid(row=4, column=1, sticky=tk.W)

        self.eliminar_btn = ttk.Button(self.frame, text="Eliminar Tarea",
command=self.eliminar_tarea)
        self.eliminar_btn.grid(row=5, column=1, sticky=tk.W)

        self.actualizar_lista()

    def agregar_tarea(self):
        titulo = self.titulo_entry.get()
        descripcion = self.descripcion_entry.get()
        try:
            self.gestor.agregar_tarea(titulo, descripcion)
            self.actualizar_lista()
        except ValueError as e:
            messagebox.showerror("Error", str(e))

    def actualizar_lista(self):
        self.tareas_listbox.delete(0, tk.END)
        for indice, tarea in enumerate(self.gestor.obtener_tareas()):
            estado = "Completada" if tarea.completada else "Pendiente"
            self.tareas_listbox.insert(tk.END, f"{indice + 1}. {tarea.titulo} - {estado}")

    def marcar_completada(self):
        seleccion = self.tareas_listbox.curselection()
        if seleccion:
            indice = seleccion[0]
            self.gestor.marcar_completada(indice)
            self.actualizar_lista()
        else:
            messagebox.showwarning("Advertencia", "Selecciona una tarea para marcar como
completada")

    def eliminar_tarea(self):
        seleccion = self.tareas_listbox.curselection()
        if seleccion:
            indice = seleccion[0]
            self.gestor.eliminar_tarea(indice)
            self.actualizar_lista()
        else:
            messagebox.showwarning("Advertencia", "Selecciona una tarea para eliminar")

def run():
    root = tk.Tk()
    gestor = GestorTareas()
    app = GestorTareasGUI(root, gestor)
    root.mainloop()

if __name__ == "__main__":
    run()
```

Pruebas Unitarias (tests/test_gestor_tareas.py)

```
import unittest
from gestor_tareas import GestorTareas
```

```
class TestGestorTareas(unittest.TestCase):
    def setUp(self):
        self.gestor = GestorTareas()

    def test_agregar_tarea(self):
        self.gestor.agregar_tarea("Tarea 1", "Descripción de la tarea 1")
        self.assertEqual(len(self.gestor.tareas), 1)
        self.assertEqual(self.gestor.tareas[0].titulo, "Tarea 1")
        self.assertEqual(self.gestor.tareas[0].descripcion, "Descripción de la tarea 1")

    def test_agregar_tarea_sin_titulo(self):
        with self.assertRaises(ValueError):
            self.gestor.agregar_tarea("", "Descripción")

    def test_marcar_completada(self):
        self.gestor.agregar_tarea("Tarea 1", "Descripción de la tarea 1")
        self.gestor.marcar_completada(0)
        self.assertTrue(self.gestor.tareas[0].completada)

    def test_eliminar_tarea(self):
        self.gestor.agregar_tarea("Tarea 1", "Descripción de la tarea 1")
        self.gestor.eliminar_tarea(0)
        self.assertEqual(len(self.gestor.tareas), 0)

if __name__ == "__main__":
    unittest.main()
```

README.md:

```
# Gestor de Tareas

Esta es una aplicación de gestión de tareas desarrollada en Python. Permite agregar, ver, marcar
como completadas y eliminar tareas. La aplicación utiliza Tkinter para la interfaz gráfica y sigue
el enfoque de Desarrollo Guiado por Pruebas (TDD) junto con GitFlow para la gestión del código.

## Requisitos

- Python 3.x
- Tkinter (incluido en la instalación estándar de Python)

## Instalación

1. Clona este repositorio:
   ```bash
   git clone <URL_del_repositorio>```
```
```