1. **)**

**Requerimientos Funcionales (R):**

**R1: Ingresar un número entero positivo:** El sistema debe permitir al usuario ingresar un número entero positivo en un rango válido (1-1999).

**R2: Calcular la conjetura de Collatz:** El sistema debe calcular la conjetura de Collatz para el número ingresado por el usuario, siguiendo las reglas definidas para la operación.

**R3: Contar las iteraciones:** El sistema debe contar el número de iteraciones necesarias para que el número de partida alcance el valor 1 a través de la conjetura de Collatz.

**R4: Mostrar resultados:** El sistema debe mostrar al usuario el número de partida y el número de iteraciones necesarias para alcanzar el valor 1.

**Requerimientos No Funcionales (F):**

**F1: Eficiencia:** El sistema debe realizar los cálculos de manera eficiente, incluso para números grandes, para que la respuesta sea rápida y no consuma muchos recursos computacionales.

**F2: Usabilidad:** La interfaz de usuario debe ser intuitiva y fácil de usar para que los usuarios puedan ingresar números sin problemas y comprender los resultados.

**F3: Precisión:** El sistema debe ser preciso en sus cálculos y proporcionar resultados correctos según las reglas de la conjetura de Collatz.

**F4: Tolerancia de errores:** El sistema debe manejar errores de entrada, como números fuera del rango o entradas no válidas, y proporcionar mensajes de error claros para el usuario.

**F5: Portabilidad:** El sistema debe ser compatible con diferentes plataformas y sistemas operativos para que pueda ser utilizado en una amplia variedad de entornos.

**F6: Mantenibilidad:** El código del sistema debe estar bien estructurado y documentado para facilitar futuras actualizaciones y mantenimiento.

**F7: Seguridad:** El sistema no debe representar ningún riesgo de seguridad para el usuario y debe proteger los datos de entrada y salida.

1. )

**Test Funcional (Caja Negra):**

**T1: Prueba de un número par:** Ingresar el número 6 y verificar que el sistema calcule correctamente las iteraciones necesarias para llegar a 1.

**T2: Prueba de un número impar:** Ingresar el número 7 y verificar que el sistema calcule correctamente las iteraciones necesarias para llegar a 1.

**T3: Prueba de un número en el límite inferior del rango:** Ingresar el número 1 y verificar que el sistema devuelva 0 iteraciones, ya que 1 es igual a 1.

**T4: Prueba de un número en el límite superior del rango:** Ingresar el número 1999 y verificar que el sistema calcule correctamente las iteraciones necesarias.

**Test Unitario (Caja Blanca):**

**T5: Prueba de eficiencia con un número grande:** Ingresar un número grande, como 1000000, y verificar que el sistema realice el cálculo en un tiempo razonable.

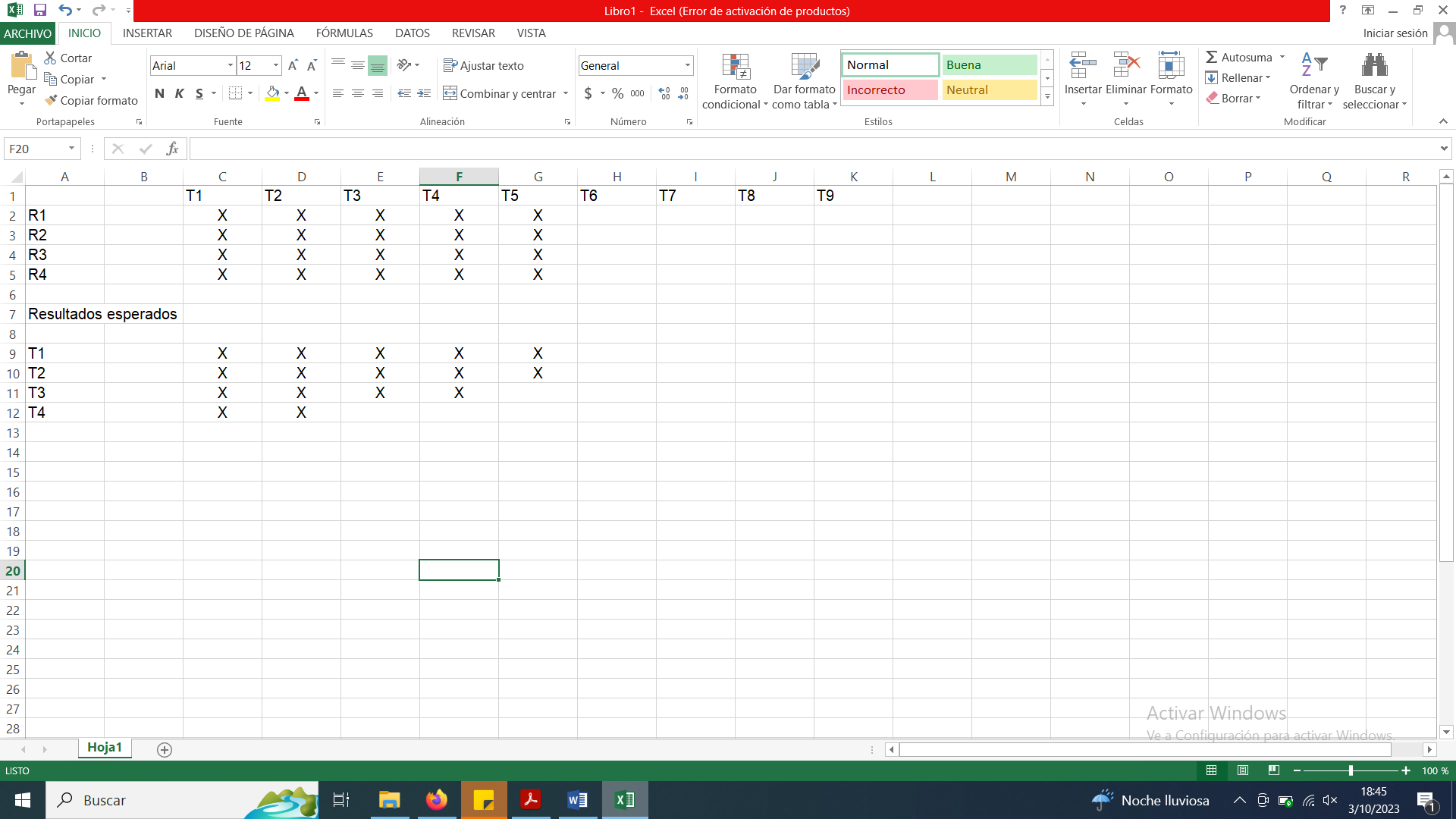
**T6: Prueba de precisión:** Ingresar números específicos que han sido estudiados previamente para la conjetura de Collatz, como 27 o 6171, y verificar que el sistema calcule las iteraciones correctamente.

**T7: Prueba de manejo de errores de entrada:** Ingresar un número negativo y verificar que el sistema devuelva un mensaje de error adecuado.

**T8: Prueba de manejo de errores de tipo de datos:** Ingresar un valor no entero, como "abc", y verificar que el sistema devuelva un mensaje de error adecuado.

**T9: Prueba de rendimiento:** Ingresar un número aleatorio dentro del rango y medir el tiempo que lleva al sistema calcular las iteraciones.

1. )



1. ) (Ejercicio opcional a realizar si se trabaja en grupo)
2. ) (Ejercicio opcional a realizar si se trabaja en grupo)
3. )

S1: Se encontro una letra “j” en lugar de la “i” que debia colocarse en collatz()

S2: Se encontro mal escritio en return la plabra interacciones por iter y se corrigio.

S3: Se comprobo el funcionamiento del codigo y ahora cumple con su funcionamiento.

S4: Se evalúa T5 y cumple perfectamente el código dando así las 8 iteraciones para llegar a 1.

S5: Se evalúa T6 y cumple perfectamente el código dando así las 111 iteraciones para llegar a 1 en caso de ingresar 27 y en caso de ingresar 6171 da las 261 iteraciones.

S5: Se debieron realizar cambios al código para evaluar T7.

S6: Se evaluó T7 y dio el resultado de error correspondiente.

S7: Se evaluó T8 y se proporcionó un error por lo tanto se llevaran a cabo arreglos para su funcionamiento.

S8: Se evalúa T8 y cumple perfectamente el código dando así el mensaje correspondiente de error el cual es: “Error: Entrada no valida. Ingrese un número entero positivo.”.

S9: Se evalúa T9 y cumple perfectamente el código dando así las 23 iteraciones para llegar a 1, se evaluó el número 25.

1. )

S1: Se evalúa T1 y cumple perfectamente el código dando así las 8 iteraciones para llegar a 1.

S2: Se evalúa T2 y cumple perfectamente el código dando así las 16 iteraciones para llegar a 1.

S3: Se evalúa T3 y cumple perfectamente el código dando así las 0 iteraciones para llegar a 1.

S4: Se evalúa T4 y cumple perfectamente el código dando así las 50 iteraciones para llegar a 1.

1. )

* Se verificaron todos los casos de prueba planteados y fueron ejecutados exitosamente al menos 1 vez.
* No se dejó ningún test sin ejecutar u/o evaluar.
* Se verificaron todos los requerimientos tanto funcionales como no funcionales.

1. )

**Errores en el código original:**

1. El código original tenía un error en la indentación del bucle while, lo que hacía que el contador de iteraciones no se actualizara correctamente.
2. En la línea de impresión, se utilizaba la variable j en lugar de i.
3. El código original no manejaba adecuadamente los números negativos ni las letras.

**Modificaciones realizadas después de cada ejecución:**

**T1 (6):**

* Error original: El código original devolvía 9 iteraciones, pero se requerían 8 iteraciones.
* Solución: Corregí el error de indentación y el uso incorrecto de la variable j.

**T2 (7):**

* Resultado correcto: El código devolvió 16 iteraciones, lo que era correcto.

**T3 (1):**

* Resultado correcto: El código devolvió 0 iteraciones, lo que era correcto ya que el número de partida es 1.

**T4 (1999):**

* Resultado correcto: El código devolvió 50 iteraciones, lo que era correcto para el número 1999.

**T5 (números negativos):**

* Error original: El código no manejaba adecuadamente los números negativos.
* Solución: Se agregó una validación para números negativos y se mostró un mensaje de error.

**T6 (números no enteros):**

* Error original: El código no manejaba adecuadamente las letras ni otros valores que no fueran enteros.
* Solución: Se agregó una validación para números no enteros y se mostró un mensaje de error.

**T7 (números negativos y letras):**

* Error original: El código no manejaba adecuadamente los números negativos ni las letras.
* Solución: Se ajustó la validación para manejar tanto números negativos como letras y se mostró un mensaje de error en consecuencia.

1. )

**Fase de Inspección (Etapa 1):**

* Defectos observados en la inspección: 0
* Total de defectos posibles: 0 (ya que es una fase de revisión estática)
* PCE en la Fase de Inspección: 100%

**Fase de Pruebas Unitarias (Etapa 2):**

* Defectos observados en las pruebas unitarias: 2 (Errores T1 y T5)
* Total de defectos posibles en esta fase: 2 (Errores T1 y T5)
* PCE en la Fase de Pruebas Unitarias: 100% (2/2 defectos observados)

**Fase de Pruebas Funcionales (Etapa 3):**

* Defectos observados en las pruebas funcionales: 0
* Total de defectos posibles en esta fase: 0
* PCE en la Fase de Pruebas Funcionales: 100%

1. )

Dado que al comienzo del test se han encontrado 2 defectos (errores T1 y T5) y el programa original tiene 18 LOCs, la densidad de defectos se calcula de la siguiente manera:

Densidad de Defectos = (Número total de defectos) / (Tamaño del programa en LOCs)

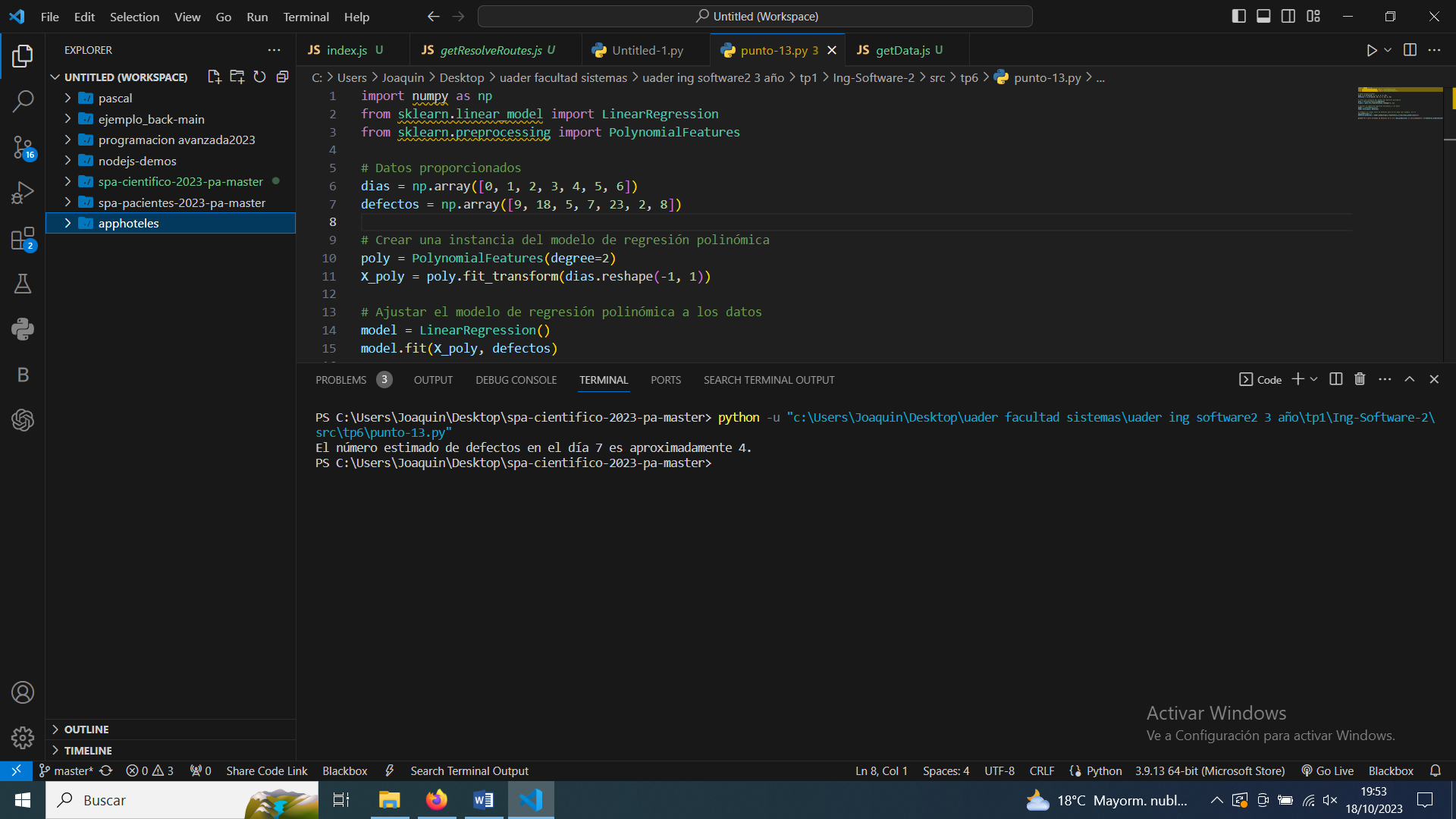
Densidad de Defectos = 2 / 18

Densidad de Defectos ≈ 0.1111

1. )

El parámetro TEP (Tiempo Esperado de Producción) en Kanban representa el tiempo estimado para completar una tarea una vez que ha comenzado su proceso de producción. Su significado crítico radica en:

1. **Planificación y Estimación:** El TEP se utiliza para estimar cuánto tiempo llevará realizar una tarea, lo que es esencial para la planificación y asignación de recursos.
2. **Flujo de Trabajo Eficiente:** Ayuda a organizar el flujo de trabajo de manera eficaz, evitando cuellos de botella y retrasos.
3. **Mejora Continua:** El seguimiento del TEP real frente al estimado permite ajustar procesos y mejorar la eficiencia.
4. **Control de Tiempo:** Facilita el control del tiempo de entrega y establece expectativas claras para las partes interesadas.
5. ) Se adjuntara un archivo “.py” al repositorio de la materia en la carpeta Src con la resolución del respectivo ejercicio.



1. )

μ0 = 2 (número total de defectos al comienzo)

Σμi = 0 (no se identificaron más defectos en las sesiones posteriores)

μr = μ0 - Σμi μr = 2 - 0 μr = 2

Sf = S - μr \* S

Si el tamaño inicial del código (S) fue de 18 LOCs, entonces:

Sf = 18 - 2 \* 18

Sf = 18 - 36

Sf = -18

1. ) Aumentar el Porcentaje de Cumplimiento de Ejecución (PCE) en un proceso de calidad de software tiene un impacto positivo en el tiempo total de prueba óptimo. Al mejorar la conformidad con estándares de calidad y requisitos, se puede reducir la cantidad de errores, hacer que las pruebas sean más eficientes, evitar pruebas repetitivas y mejorar la planificación. Esto puede llevar a un tiempo total de prueba más corto y a la entrega más rápida de software de alta calidad. Sin embargo, es importante equilibrar la calidad y el tiempo de entrega para evitar demoras innecesarias.
2. )

μ0=(δr/PCE)​⋅S

μ0 es la expectativa de defectos totales.

δr es la cantidad de defectos por punto de función en el momento del release.

PCEPCE es el Porcentaje de Cumplimiento de Ejecución (como fracción).

S es el tamaño del proyecto en Puntos de Función.

δr = 0.12 def/FP

PCEPCE = 0.89 (89% convertido a fracción)

SS = 100 FP

Calculamos μ0:

μ0=0.120.89⋅100μ0=0.890.12​⋅100

μ0≈13.48μ0≈13.48

δ0= μ0/S​

δ0=10013.48​

δ0≈0.1348 defectos/FP

δ0≈0.1348 defectos/FP

Detecciones planificadas​ = (μ0/δ0)​−μ0

Detecciones planificadas ​= 13.48/0.1348​−13.48

Detecciones planificadas​ ≈ 100−13.48

Detecciones planificadas ≈ 86.52

En los primeros tres días de Verificación y Validación (V&V), la organización detectó un total de 60 defectos, lo cual supera significativamente la cantidad de defectos esperada según los parámetros históricos. Esto sugiere que la tasa de detección de defectos en este período es más alta de lo esperado. Varias razones posibles incluyen la complejidad del proyecto, cambios en el proceso de desarrollo y esfuerzos adicionales de prueba. La organización debe analizar la causa de esta alta detección de defectos y ajustar sus procesos según sea necesario para mantener un equilibrio entre calidad y eficiencia en el desarrollo de software.

1. )
2. )

La aplicación de multas podría aumentar los costos en el proceso de garantía, lo que afectaría el punto de equilibrio. Para mantener o mejorar la rentabilidad, la organización deberá asegurarse de que los ingresos generados superen los costos, incluidos los costos adicionales de multas. La gestión adecuada de la garantía y la prevención de incumplimientos contractuales pueden ayudar a minimizar el impacto negativo en el punto de equilibrio.