

Ingeniería del Software

Práctica 4

Implementación y pruebas

2º Grado en Ingeniería Informática
Universidad de Córdoba
Curso 2024/25

1. Organización de la práctica

2. Scrum

3. Eclipse

4. Pruebas unitarias

1. Organización de la práctica

Sesiones

- Sesión 1. Implementación.
 - Utilizar C++ como lenguaje y Eclipse como IDE.
 - Implementación siguiendo metodología basada en Scrum.
- Sesión 2. Pruebas unitarias.
 - Plugin *CUTE* de Eclipse.

1. Organización de la práctica

Semanas de trabajo

- 4 semanas de trabajo en total para completar la documentación.
- La fecha límite (no prorrogable) para la entrega de la documentación:
 - **20 diciembre**
- Sprints semanales:

Sprint	Semana
Sprint 1	Semana del 25 al 29 de noviembre
Sprint 2	Semana del 2 al 6 de diciembre
Sprint 3	Semana del 9 al 13 de diciembre
Sprint 4	Semana del 16 al 20 de diciembre

1. Organización de la práctica

Semanas de trabajo

- Se evaluará la distribución y planificación semanal del trabajo.
- Reflejado en el repositorio en **GitHub** y en *YouTrack o Trello*.
- No valdrá con subirlo todo el último día, debe seguirse la metodología basada en Scrum y la organización en *sprints*.

1. Organización de la práctica

Documentación

- **Manual técnico**

- Introducción y planificación
- Análisis de requisitos (especificación de requisitos, historias de usuario, y casos de uso), opcional IEEE830
- Diseño (diagrama de clases y diagramas de secuencia)
- Matrices de validación
- **Implementación**
- **Pruebas**

- **Manual de usuario (incluyendo video tutorial demostrativo)**

1. Organización de la práctica

Documentación

- Implementación (especificaciones)
 - Enlace al repositorio donde se aloja el código
 - Entorno y herramientas de trabajo (IDE, lenguaje de programación...)
 - Qué historias se han seleccionado en cada sprint
 - Cómo se han priorizado
 - Decisiones de implementación (arquitectura, interfaz, datos, procedimientos...)

1. Organización de la práctica

Documentación

- Pruebas
 - Diseño de las pruebas
 - Resultado (ver plantilla Moodle)
 - Errores detectados y cómo se han solucionado (ver plantilla Moodle)
 - Código debe estar alojado en GitHub también

1. Organización de la práctica

Documentación

- Manual de usuario (independiente al manual técnico)
 - El manual de usuario es el documento que permite a las personas que utilizan los sistemas de información su entendimiento y uso de las funcionalidades que este posee. Además, es una guía de asistencia para el usuario final sobre el funcionamiento de los aplicativos y de solución a los problemas más comunes.

<https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/465>

- Video tutorial demostrativo en el repositorio Github

1. Organización de la práctica

2. Scrum

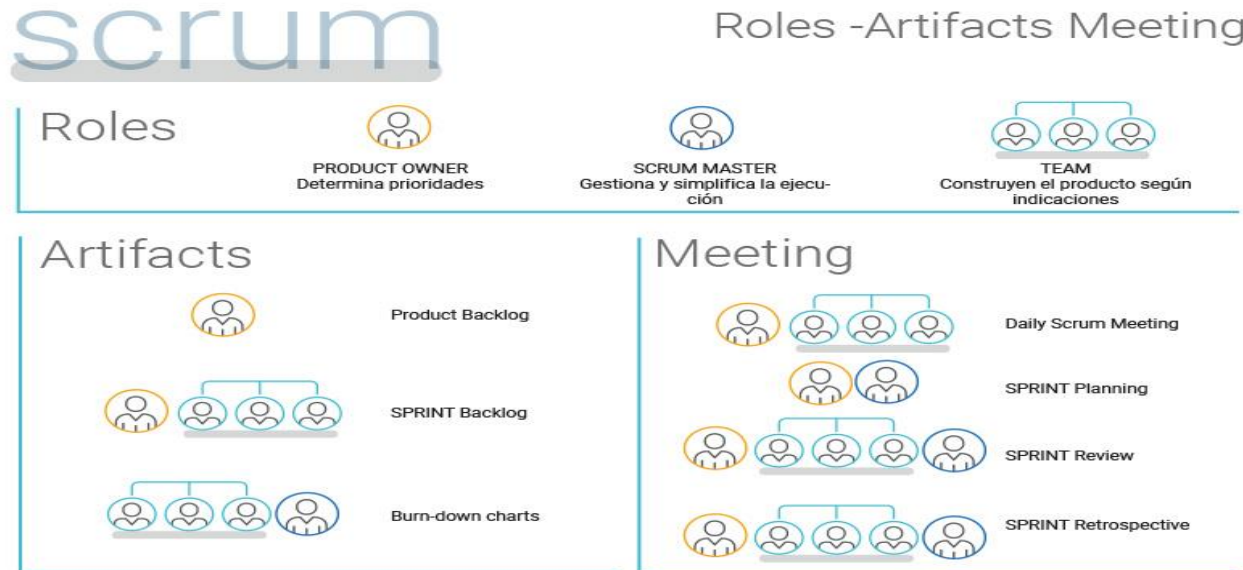
3. Eclipse

4. Pruebas unitarias

2. Scrum

Aspectos de la metodología

- Cada *sprint* tendrá una duración de una semana.
- La planificación en *sprints* debe quedar reflejada en *YouTrack o Trello*.

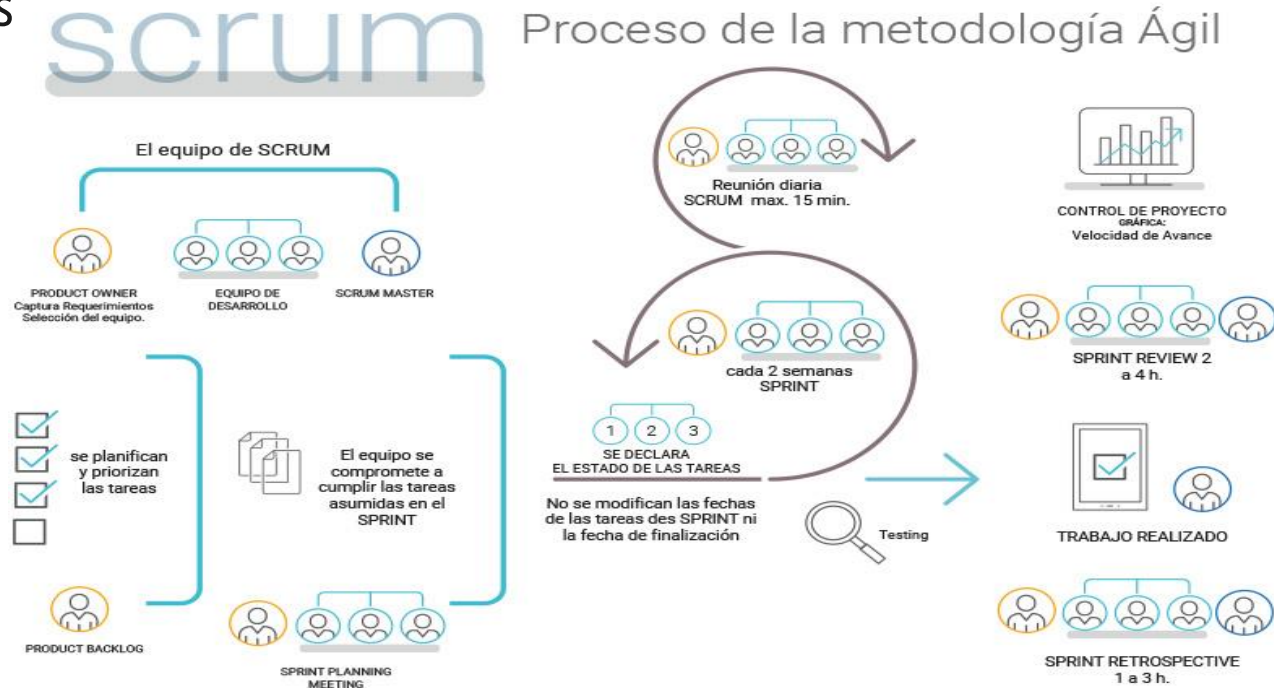


Nuestra esencia consiste en la división del trabajo total (Product Backlog) en periodos cortos de tiempo (1-4 semanas) los cuales son llamados SPRINTS, interacciones continuas de desarrollo que facilita la introducción de modificaciones "sobre la marcha", mejorando continuamente el proceso.

2. Scrum

Aspectos de la metodología

- En cada *sprint*, uno de los miembros del equipo debe actuar como **Product owner**.
- El rol será rotatorio.
- Cada miembro debe serlo al menos una vez.



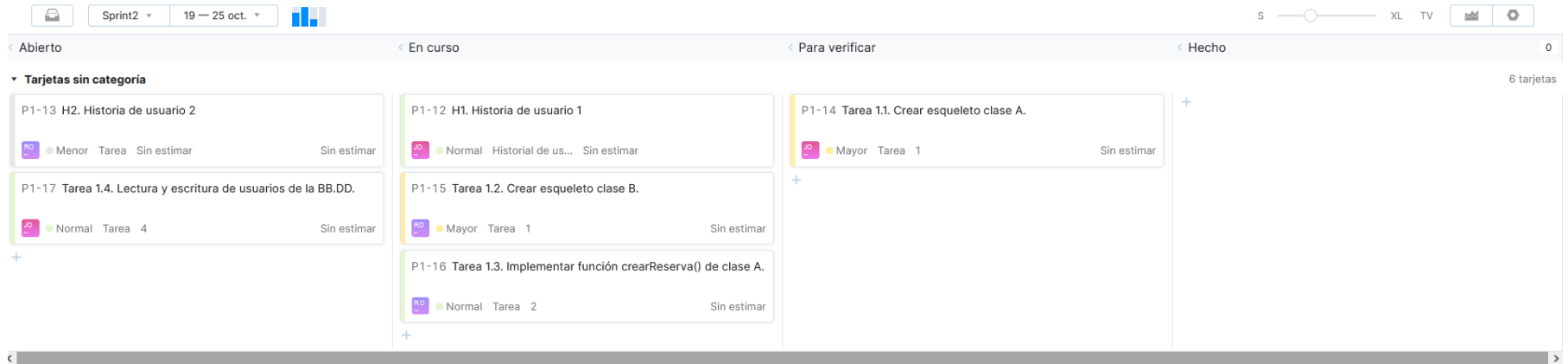
2. Scrum

Aspectos de la metodología

- El *Product owner* es el encargado de seleccionar las historias de usuario a completar durante el *sprint*.
 - Especificar los requisitos asociados a la historia.
- Cada historia se divide en tareas, y se les asigna responsable, tiempo estimado (en días), y prioridad.
 - Las tareas deben ser abordables a lo largo del sprint.
- Debe quedar reflejado en *YouTrack* o *Trello*.
- La responsabilidad recae en el *Product owner*, pero el resto del equipo podrá participar en la organización y distribución del trabajo.

2. Scrum

Aspectos de la metodología



2. Scrum

Aspectos de la metodología

>_

...

P1-14 Creada por root Hace 14 minutos
Actualizado por root Hace 7 minutos

Visible para lectores de incidencias ▾

★

Tarea 1.1. Crear esqueleto clase A.

Texto normal ▾ | **B** *I* S ...

Visual Markdown Aa

Historia de usuario: H1.

Requisitos involucrados: RF1, RF3.

Descripción de la tarea: ...

Haga clic para adjuntar, arrastre archivos o pulse Ctrl+V para pegar una imagen

Generar URL de plantilla de incidencia

Aplicar cambios

Cancelar

Proyecto Proyecto1

Prioridad Mayor

Tipo Tarea

Estado Para verificar

Usuario asignado Jose

Sprints Sprint2

Ideal days Sin estimar

Original estimation 1

Observadores 2 > ★ Dejar de observar

Paneles > + Agregar al panel

2. Scrum

Aspectos de la metodología

- Eventos
 - **Planificación del sprint.** El *producto owner* selecciona la funcionalidad que se va a incorporar durante el *sprint*, y cómo se realizará.
 - **Reunión diaria** (idealmente). Distribuir tareas y/o evaluar el progreso. En nuestro caso, podría tenerla el *producto owner* por separado con el resto del equipo si es necesario.
 - **Revisión del sprint.** Reunión informal entre el equipo y el cliente donde se muestra el funcionamiento y/o estado del software hasta el momento.

2. Scrum

Aspectos de la metodología

★ 17/11/21. Planificación Sprint X ...

Product Owner: Jose Moyano.

La planificación del sprint X se realiza el día ... a la hora ...

El product owner escoge la historia de usuario HX para ser implementada a lo largo de este sprint.

Se decide dividir la historia en X tareas: tarea1, tarea2, ...

- MiembroA se encargará de ...
- MiembroB se encargará de ...

Se acuerda que el PO contactará con los distintos miembros del equipo cada dos días para conocer el estado de sus tareas.

2. Scrum

Aspectos de la metodología

★ 19/11/21. Reunión diaria. ...

El product owner contacta con MiembroA y MiembroB de forma conjunta para conocer el estado de sus tareas. Las tareas de ambos están avanzadas, y ya han implementado tarea1 y tarea2. El progreso es correcto.

Por otro lado, se reúne con MiembroC, el cual se encuentra atascado con la tarea3. Se acuerda que el PO le ayudará a solucionar sus problemas con dicha tarea para poder finalizar el sprint según esperado.

★ 23/11/21. Revisión del sprint X ...

Se han completado con éxito todas las tareas asignadas para este sprint, excepto la tareaZ. En la tareaZ falta ...

De momento, el funcionamiento del sistema es el siguiente:

- ...

Se adjunta a continuación capturas de la aplicación.

2. Scrum

Aspectos de la metodología

- El código se alojará en un repositorio Git en GitHub.
- Servirá para comprobar el trabajo semanal.
- Repositorio privado.
- Dar acceso al profesor encargado del grupo de prácticas.
 - Luis Martínez (in1macal@uco.es).

2. Scrum

Aspectos de la metodología

- Al finalizar cada *sprint*, debe haber un producto que pudiera ser entregable, aunque la funcionalidad sea reducida.
- Cada equipo debería implementar tantas historias de usuario como miembros del equipo.
- La aplicación debería tener un menú principal donde se puedan seleccionar las distintas funcionalidades implementadas.

1. Organización de la práctica

2. Scrum

3. Eclipse

4. Pruebas unitarias

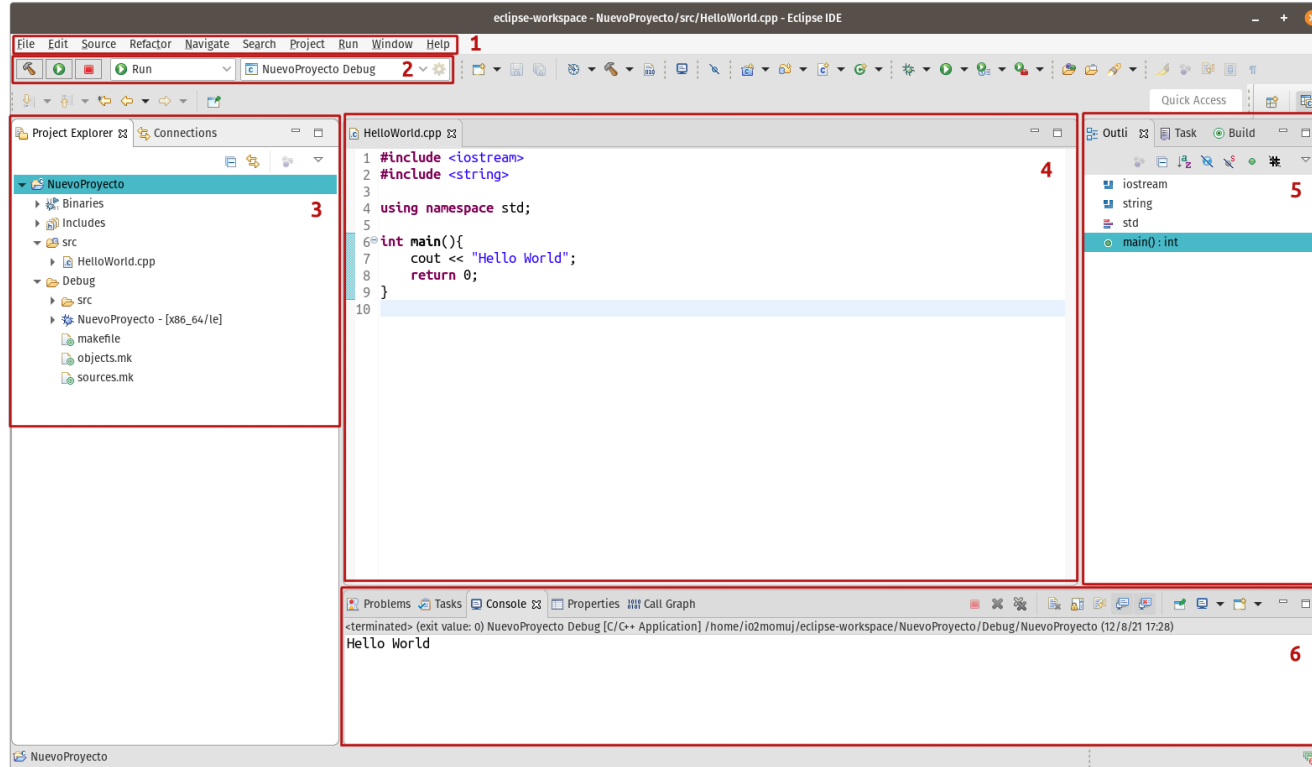
3. Eclipse

Introducción

- Entorno de desarrollo integrado (IDE).
- Originalmente para Java; disponible para C/C++.
- Permite instalar extensiones.
- Integrarse con Git.

3. Eclipse

Pantalla principal



3. Eclipse

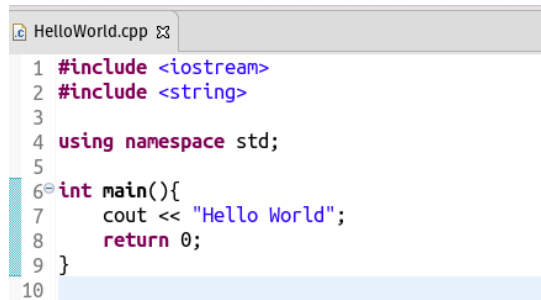
Crear nuevo proyecto

1. *File → New → C/C++ Project*
2. Escoger plantilla *C++ Managed Build*
3. Introducir nombre al proyecto. Tipo *Empty project*
4. Finalizar

3. Eclipse

Crear ejemplo simple

1. Seleccionar proyecto, y con botón derecho
 - *New* → *New source folder*
2. Introducir nombre *src*. Finalizar
3. Seleccionar el paquete creado con botón derecho
 - *New* → *File*
4. Introducir nombre para el fichero, y finalizar.
5. Escribir código en el editor.



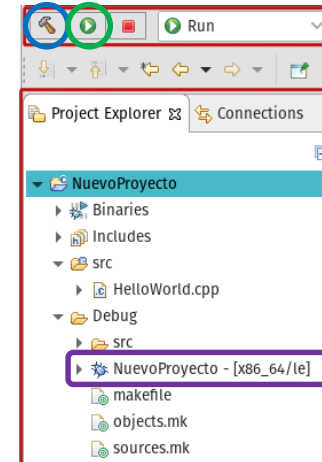
The screenshot shows a code editor window titled 'HelloWorld.cpp'. The code is as follows:

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main(){
7     cout << "Hello World";
8     return 0;
9 }
10
```

3. Eclipse


Ejecución

- Si el código tiene errores de sintaxis, Eclipse los marca.
- Para compilar, pulsar botón *Build*.
 - Se muestra en la consola el estado de la compilación.
- Aparecerá en *Project explorer* una carpeta llamada *Debug* con el ejecutable.
- Ejecutar pulsando el botón *Launch*.



3. Eclipse

Depurador

- Introducir *breakpoint* haciendo doble click junto al número de línea.
- Ejecutar con el depurador 
- Donde estaba *Outline* aparecen nuevas pestañas con el valor de las variables
- Nuevos botones para ejecutar línea a línea, etc.

1. Organización de la práctica

2. Scrum

3. Eclipse

4. Pruebas unitarias

4. Pruebas unitarias

Conceptos básicos

- Pequeño código que invoca a una parte del software a probar.
- Comprueba un supuesto cuyo resultado conocemos a priori.
- Objetivo invocar una única funcionalidad
 - Mediante un método independiente que devuelve un resultado o hace pequeña modificación.

4. Pruebas unitarias

Conceptos básicos

- Buen conjunto de pruebas:
 - Detecta alto número de errores.
 - Intenta alcanzar todos los “caminos” posibles.
- No se trata de no cometer errores de sintaxis, sino comprobar que los métodos funcionan como se esperan ante una variedad de datos de entrada.

4. Pruebas unitarias

Conceptos básicos

- Para considerarse una prueba unitaria:
 - Fácil de implementar, automatizable y repetible.
 - Probar un escenario relevante, y ser consistente en su resultado.
 - Independiente de otras pruebas unitarias.
 - Si falla, debe ser fácil detectar la causa y cómo puede solucionarse.
- Deben ser **simples** y **legibles**.
- Por lo general, utilizar un único aserto.

4. Pruebas unitarias

Diseño y codificación

- Pensar qué valores de los parámetros son más propensos a generar fallos.
- Tener en cuenta los “caminos” del programa para intentar alcanzar la mayor cobertura posible.
 - Una cobertura del 100% no suele ser posible en sistemas complejos, por lo que es importante diseñar un conjunto de escenarios lo más diverso posible.

4. Pruebas unitarias

Diseño y codificación

- Para escribir una prueba, se deben seguir los siguientes pasos:
 - Crear y configurar los objetos necesarios para ejecutar la prueba.
 - Actuar sobre un objeto para probar una funcionalidad, normalmente invocar a un método con unos parámetros escogidos.
 - Comprobar con un aserto que el resultado es el esperado.

4. Pruebas unitarias

Diseño de pruebas con CUTE

```
1  class Calculadora{
2  public:
3      int suma(int a, int b){
4          return a+b;
5      }
6
7      bool esPar(int numero){
8          if((numero % 2) == true)
9              return true;
10         else
11             return true; // Esto es un error
12     }
13 };
```

4. Pruebas unitarias

Diseño de pruebas con CUTE

```
1  void testSuma(){
2      // Inicializar objeto a probar
3      Calculadora calc = Calculadora();
4      // Valores a probar
5      int a = 5;
6      int b = 2;
7      // Obtener resultado actual
8      int resultado = calc.suma(a,b);
9
10     // Comprobar el resultado
11     ASSERT_EQUAL(7, resultado);
12 }
```

4. Pruebas unitarias

Diseño de pruebas con CUTE

```
1 void testNumeroEsPar(){
2     int numero = 2;
3     Calculadora calc = Calculadora();
4     bool resultado = calc.esPar(numero);
5     ASSERT(resultado == true);
6 }
7
8 void testNumeroEsImpar(){
9     int numero = 3;
10    Calculadora calc = Calculadora();
11    bool resultado = calc.esPar(numero);
12    ASSERT(resultado == false);
13 }
```

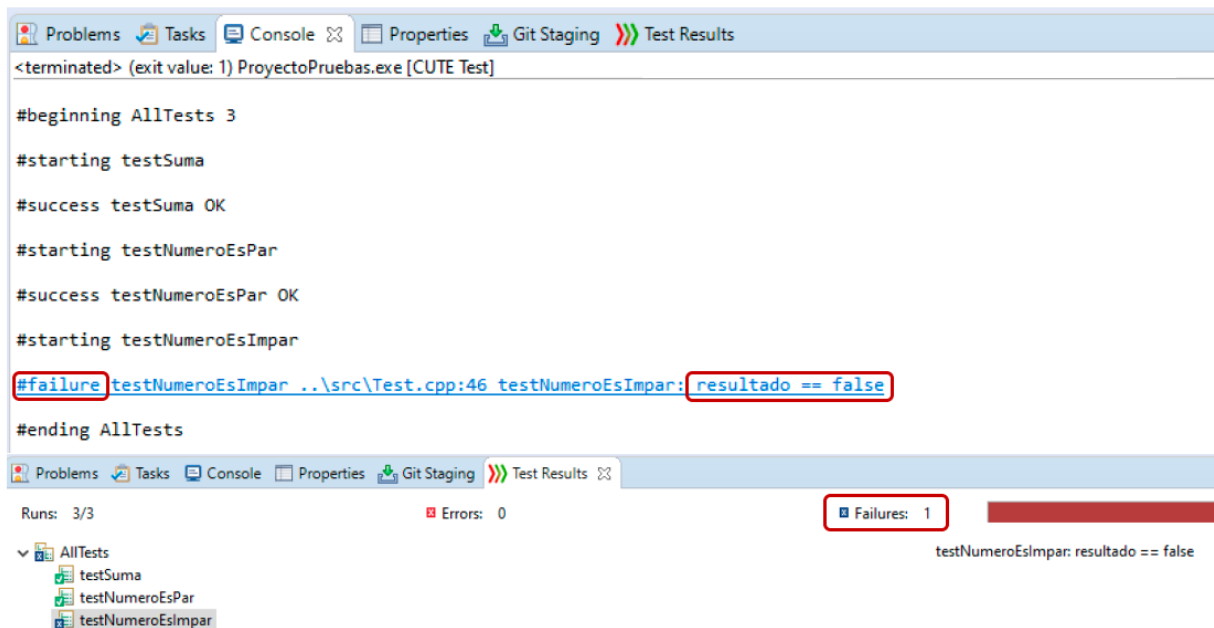
4. Pruebas unitarias

Diseño de pruebas con CUTE

```
1  bool runAllTests(int argc, char const *argv[]) {
2      cute::suite s { };
3
4      // Registrar las pruebas unitarias
5      s.push_back(CUTE(testSuma));
6      s.push_back(CUTE(testNumeroEsPar));
7      s.push_back(CUTE(testNumeroEsImpar));
8
9      // El resto de la funcion tal y como aparece por defecto
10     cute::xml_file_opener xmlfile(argc, argv);
11     // ...
12 }
```

4. Pruebas unitarias

Diseño de pruebas con CUTE



The screenshot shows the Visual Studio Test Results window for a project named 'ProyectoPruebas.exe [CUTE Test]'. The console output displays the execution of three tests: 'testSuma' (passed), 'testNumeroEsPar' (passed), and 'testNumeroEsImpar' (failed). The failure message for 'testNumeroEsImpar' is highlighted with a red box, showing the assertion 'resultado == false' failed. Below the console output, the 'Test Results' tab shows a summary: 'Runs: 3/3', 'Errors: 0', and 'Failures: 1'. A red progress bar indicates the failure. The test results tree on the left shows 'AllTests' expanded, with 'testSuma', 'testNumeroEsPar', and 'testNumeroEsImpar' listed. The 'testNumeroEsImpar' test is highlighted, and its failure message 'testNumeroEsImpar: resultado == false' is displayed on the right.

```
<terminated> (exit value: 1) ProyectoPruebas.exe [CUTE Test]

#beginning AllTests 3

#starting testSuma

#success testSuma OK

#starting testNumeroEsPar

#success testNumeroEsPar OK

#starting testNumeroEsImpar

#failure testNumeroEsImpar ..\src\Test.cpp:46 testNumeroEsImpar: resultado == false

#ending AllTests
```

Runs: 3/3 Errors: 0 Failures: 1

testNumeroEsImpar: resultado == false

4. Pruebas unitarias

Entregable

- Se deben diseñar 2 pruebas por cada miembro del equipo.
- Funcionalidades no triviales del código
 - Excluir por ejemplo métodos get/set.
 - Los métodos relacionados con los mensajes de los diagramas de secuencia son buenos candidatos.

Ingeniería del Software

Práctica 4

Implementación y pruebas

2º Grado en Ingeniería Informática
Universidad de Córdoba
Curso 2024/25