# Sistema Unit Test Multiplataforma

Francisco Magdaleno Garrido

**Resumen**— Este document presenta el desarrollo de un framework para la realización de Unit test en diferentes plataformas de test. El proyecto esta enfocado a poder realizar Unit Test en diferentes plataformas como pueden ser Google Test o CppUnit y de esta forma poder reutilizar los tests creados anteriormente en caso de que se cambie de plataforma. También ofrece la posiblidad de poder ampliar y/o personalizar el framework al gusto o a la comodidad del propio usuario de una forma sencilla e intuitiva. El proyecto se ha realizado con C/C++ y con la idea de ser compatible principalmente con sistemas Linux y compiladores g++ y gcc. Actualmente se han integrado dentro de este framework las plataformas de Google Test y CppUnit, pero se ha desarrollado siempre con idea de que pueda ser ampliado a nuevas plataformas de testeo.

**Palabras clave**—Unit Test, Framework, Google Test, CppUnit, Coverage Test

**Resum**— Aquest document presenta el desenvolupament d'un framework per la realització de Unit test en diferents plataformes de test. El projecte està enfocat a poder realitzar Unit Testen diferents plataformes com poden ser Google Test o CppUnit i així poder reutilitzar els tests creats anteriorment en cas de que es canviï de plataforma. També ofereix la possibilitat de poder ampliar i/o personalitzar el framework al gust o a la comoditat del mateix usuari d'una forma senzilla i intuïtiva. El projecte s'ha realitzat amb el llenguatge C/C++ i amb la idea de ser compatible principalment amb sistemes Linux i compiladors G++ i GCC. Actualment s'han integrat dins d'aquest framework les plataformes de Google Test y CppUnit, però s'ha desenvolupat sempre amb la idea de poder ser ampliat a noves plataformes de testeig.

**Paraules clau**— Unit Test, Framework, Google Test, CppUnit, Coverage Test

**Abstract**— This document presents the development of a framework for the realization of Unit test in different test platforms. The project is focused on being able to carry out Unit Test in different platforms such as Google Test or CppUnit and in this way, to be able to reuse the tests previously created in case of a change of platform. It also offers the possibility to expand and / or customize the framework to the taste or comfort of the user in a simple and intuitive way. The project has been done with C / C ++ and with the idea of being compatible mainly with Linux systems and g ++ and gcc compilers. Currently, the Google Test and CppUnit platforms have been integrated into this framework, but it has always been developed with the idea that it can be extended to new testing platforms.

**Index Terms**— Unit Test, Framework, Google Test, CppUnit, Coverage Test

## 1 INTRODUCCIÓN

EL trabajo de fin de grado pretende poner en practicas conocimientos y competencias adquiridas durante la realización del grado. En concreto, la mención de software pretende conseguir lo siguiente: creación de aplicaciones que satisfaga las necesidades del usuario, análisis de las aplicaciones ya existentes en el ámbito del proyecto a realizar, dar solución a los problemas de estas aplicaciones existentes, crear software de calidad, crear código reusable, documentar correctamente la aplicación creada, etc.

Hoy en dia, en muchos ámbitos de la vida se utilizan sistemas informáticos, los cuales funionan gracias a un código que les da comportamiento. Esta tendencia va a más y estos sistemas necesitan ser testeados antes de ponerse a disposición de las personas para su uso. Además de ser testeadas, hay ámbitos en los que no solo necesita que el sistema sea testeado en sus funcionalidades básicas, debido a que se trata de software critico, es decir, software que en caso de fallo las consecuencias podrían ser catastróficas. Un ejemplo de estos softwares críticos són los sistemas en el ámbito de la salut, la ciencia en general o el transporte entre otros.

Hoy en dia existen sistemas de testeo, como veremos más adelante, que permiten realizar test sobre estos sistemas mencionados pero que, sin embargo, estan limitados al funcionamiento dentro de su propio entorno.

Así pues, si mezclamos los dos puntos del objetivo de la creación de software y la importancia del testeo, puede surgir un proyecto como este, en el que se intentará crear una solución para poder realizar test sobre los sistemas informáticos, pero sin perder la capacidad de ser una aplicación reusable, de calidad, etc, creando una solución que pueda ser ejecutada en diferentes plataformas de test sin necesidad de modificar el código creado para cualquiera de ellas en una primera instancia.


- *E-mail: francisco.magdalenog@gmail.com*
- *Mención: Ingeniería del Software*
- *Tutor del trabajo: Lluis Gesa Boté*
- Curso 2018/2019

# 2 OBJETIVOS

En esta sección definiremos los objetivos que se desean alcanzar al finalizar este proyecto.

## 2.1 Principales objetivos

Este proyecto tenía varios objetivos los cuales, a continuación, definiremos con una breve explicación.

El primer objetivo de este proyecto era analizar los frameworks actuales más usados en la actualidad con la finalidad de definir cuales serian los frameworks mas adecuados para integrar en la solución de nuestro proyecto.

Como segundo objetivo, una vez definidos cuales eran estos frameworks más usados, era analizar la estructura de cada uno de ellos para poder saber si una solución común era posible o no. Para ello, dediqué un tiempo a trabajar con cada uno de ellos sobre una aplicación de prueba sencilla.

En tercer lugar, en caso de haber tenido un análisis con resultado positivo, se quería crear una solución libre, con la finalidad de que cualquiera pudiera utilizarla y modificarla.

Por utlimo y en cuarto lugar, también quería que esta solución fuera compatible con entornos libres cómo puede ser en este caso el compilador estándar por excelencia para C++, GNU C++ .

## 2.2 Objetivos Secundarios

Una vez alcanzados estos objetivos principales con éxito, se buscaba plantear algún objetivo secundario con el objetivo de llevar este proyecto un poco mas lejos, a un campo más amplio.

Se planteo un objetivo secundario, el cual fue analizar cómo se resuelve el tema del Unit Test es sistemas encastados, para así adentrar el proyecto en el mundo de las IoT, un mundo que esta creciendo en gran medida en los últimos años.

# 3 ESTADO ACTUAL

Existen otras plataformas que permiten realizar Unit Test. A continuación, comentaremos brevemente algunas de ellas.

## 3.1 Google Test

Google Test[1] es una librería para realizar Unit Test sobre el lenguaje de programación C++, aunque también permite realizar pruebas sobre archivos de fuente C. Esta librería está desarrollada por Google.

## 3.2 CppUnit

CppUnit[2] es un modulo de test para el lenguaje C++, aunque igual que Google Test, permite realizar pruebas sobre archivos fuente C.

## 2.2 Boost Test

Boost Test[3] es un conjunto de librerias para realizar test sobre C++. Inicialmente fue creado por Beman Dawes y David Abrahams, aunque ha seguido creciendo gracias a la comunidad de boost.

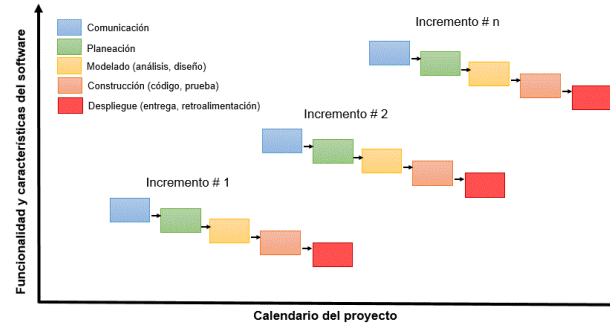
## 2.3 Comparación con el proyecto

Todas las plataformas mencionadas en los subapartados anteriores se asemejan y difieren en los mismos puntos con la solución que se quiere crear en este proyecto.

Las tres plataformas mencionadas, al igual que esta solución, permiten realizar Unit Test sobre nuestras aplicaciones, que al fin y al cabo és la razón por la que se crearon. Sin embargo, estas tres plataformas tienen un carácter "cerrado", solo puedes realizar tests sobre su estructura y código. Aquí yace la diferencia de este proyecto, el cual no busca crear una "nueva" plataforma con la que crear y realizar Unit Test si no que busca crear una solución para poder con un solo código y estructura poder ejecutarse en estas diferentes plataformas mencionadas anteriormente sin necesidad de modificaciones.

# 4 METODOLOGIA

Para la realización del proyecto se ha usado la metodología del modelo iterativo e incremental[4], un método, en mi opinión, muy adecuado para este tipo de proyectos.




*1. Modelo Incremental[5]*

Este método te permite trabajar organizadamente y por fases permitiéndote organizar claramente las tareas a realizar en cada iteración, revisar el trabajo hecho al final de cada una de estas y permitiéndole al proyecto crecer poco a poco entre cada iteración sin cambios demasiado drásticos, lo que permite identificar mas fácilmente donde y cuando se han podido generar los errores que vayan apareciendo a lo largo del proyecto. Además, es un modelo que permite una gran adaptabilidad a las necesidades

de cada proyecto y equipo.
Para realizar esta metodología he utilizado basicamentedos herramientas: Trello y Bitbucket.

Trello me ha permitido saber en todo momento las tareas que ya había realizado y estaban finalizadas, las tareas que estaba realizando en cada instante del proyecto, y aquellas tareas que estaban por realizarse en alguna de las iteraciones restantes hasta finalizar el proyecto. Esto se resume en un una modelo TO DO – DOING – DONE[6], que se compenetra muy bien con el modelo general que estaba usando en el proyecto, el modelo incremental e iterativo.

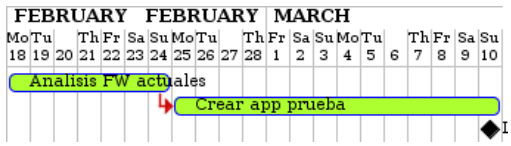
Por otro lado, tenemos a bitbucket, un repositorio online que me ha servido para ir subiendo los avances que iba haciendo en el proyecto y así tener un control de versiones, además de tener el código en un lugar seguro y evitar perdidas. Aún y asi, esta herramienta ha sido una herramienta temporal ya que la versión definitiva del proyecto se alojará en un repositorio de GitHub.

## 5 PLANIFICACIÓN

Con la metodología explicada en el anterior punto del documento, se realizó una planificación basad en diferentes sprints. A continuación, se explica en que ha consistido cada iteración dentro del proyecto y aquello que hemos extraído en cada una de ellas.

### 5.1 Iteración 1

En esta primera iteración, se llevo a cabo la investigación sobre los frameworks actuales más usados para definir que frameworks de entre todos los actuales iba a definir entre los actuales y se creo una aplicación de ejemplo sobre la que iba a trabajar con los frameworks elegido y con la solución del proyecto.



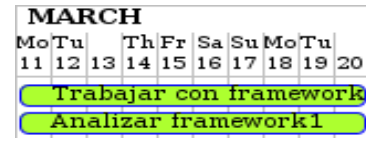

*2. Iteración 1*

Al final de esta iteración tenía definidos los frameworks que quería integrar y tenia la aplicación de ejemplo finalizada. En un primer momento los frameworks elegidos fueron Google Test, CppUnit y Boost Test.

Por otra parte, la aplicación de prueba que más adelante nos permitirá trabajar los diferentes frameworks elegidos, consiste en una calculadora sencilla la cual lee por teclado la operación deseada y la realiza. Esta calculadora soporta las operaciones de suma, resta, multiplicación, dividisión y modulo.

### 5.2 Iteración 2

En la segunda iteración, se trabajo con el primer framework de los tres elegidos, en este caso con Google Test. Con este trabajo se pretendía entender como se declara un test, como se lanza y como se escribe.

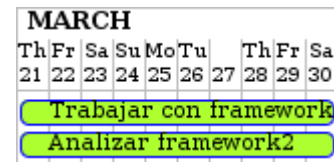



*3. Iteración 2*

Al finalizar la iteración entendia cómo funcionaba el framework y había extraído las formas de crear un test, las diferentes macros que tiene el framework para crear los asserts y la estructura.

### 5.3 Iteración 3

En esta tercera iteración, se paso a trabajar el framework numero 2, en este caso iba a ser CppUnit, pero mi desconocimiento de este hizo que por equivocación trabajara con la propia adaptación de este que incluye Microsoft Visual Studio.

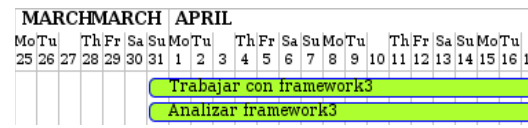



*4. Iteración 3*

Al igual que en la iteración anterior, extrajé la información de creación y macros para el posterior análisis.

### 5.4 Iteración 4

En la iteración 4 se pretendía trabajar con el tercer y ultimo framework a integrar en la solución. Debido al error que ocurrió en la iteración anterior, decidí en esta trabajar con el framework de CppUnit original y aparcar el que iba a ser trabajador en esta iteración, en este caso Boost Test, para una iteración futura si conseguia tiempo.




*5. Iteración 4*

Además de lo mencionado anteriormente, esta fue una de las iteraciones mas controversas. Al comienzo de esta, mi

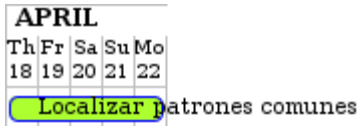
intención era trabajar este framework en el entorno de Microsoft Visual Studio, al igual que los dos anteriores pero el hecho de no estar directamente integrado en el entorno me dio diversos problemas de depuración y compilación. También tuve una segunda complicación y es que, por diversas razones, en mi dia a dia no le pude dedicar el tiempo que le había podido dedicar a las iteraciones 2 y 3.

Tras estas complicaciones, amplié el tiempo de duración de la iteración, pasando de 10 dias a 18 dias.

Finalmente, al finalizar esta iteración, extrajé la misma información que con los frameworks 1 y 2, es decir, Estructura de los tests y macros.

### 5.5 Iteración 5

Después de la iteración 4, encontramos esta breve iteración de 5 dias.




*6. Iteración 5*

Esta era una iteración breve con una sola tarea. En ella encontramos un punto de inflexión en el proyecto ya que esto definiría el porvenir de este. Esta tarea consistia en observar las diferentes estructuras de los frameworks que había trabajado en las iteraciones anteriores para identificar si la solución era posible o no y si así era por donde podía unirlos en mi solución. El resultado finalmente fue positivo, aunque decidí excluir de este el framework 2 (CppUnit de Visual Studio) debido a que es un framework unicamente para el entorno de Visual Studio causando que fuera de este, tuviéramos una opción totalmente invalida sin importar el entorno en el que nos encontraramos.

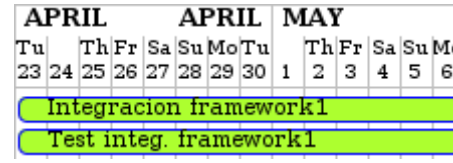
### 5.6 Tarea de documentación

En este apartado, hago mención especial a la tarea de documentación ya que no es una tarea al uso. Esta tarea no pertenece a una iteración en concreto si no que se va realizando a lo largo de diferentes iteraciones hasta el final del proyecto.
Esta documentación esta formada por dos partes distintas:

1. Wiki de GitHub: Aquí se encuentra la información relacionada con el manual de uso del framework, es decir, la instalación y preparación del entorno, cómo escribir los tests, cómo compilar y ejecutar tests, cómo realizar coverage testing y una propuesta para la ampliación del framework.
2. Documentación del código y las macros: Esta esta generada con doxygen y podemos encontrar la descripción de las diferentes funcionalidades de test, documentación de la aplicación de prueba y documentación de la estructura del framework.

### 5.7 Iteración 6

En esta iteración comienza el desarrollo de la propia iteración con la integración del primer framework, Google Test.



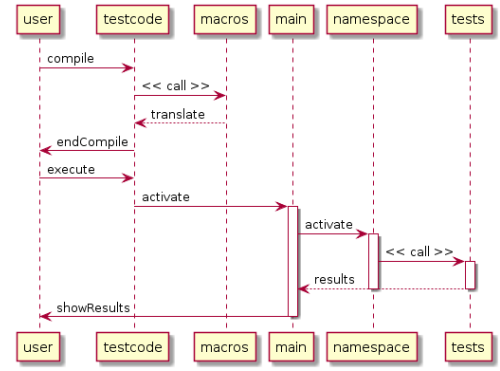

*7. Iteración 6*

Tras esta primera integración, se obtuvo la primera versión del framework. En esta primera versión ya podíamos escribir, compilar y ejecutar tests con nuestro propio framework sobre la plataforma de Google Test.

Como resultado tenia la estructura en la que se basa el framework. La estructura es la siguiente:

1. Fichero con la función main.
2. Fichero con namespace donde se definen y ejecutan los tests suite y sus respectivos tests.
3. Fichero con la implementación de los tests.
4. Fichero con las diferentes macros del framework.

Acontinuación podemos ver el la relacion entre estos 4 ficheros para la ejecución de tests.




*8. Relacion entre ficheros*

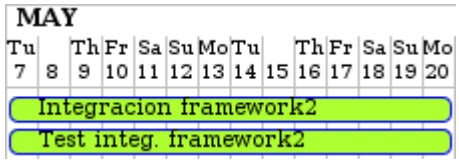
*Nota: testcode hace referencia a todo el código, la aplicación de test en general.*

Por la parte de test, una vez comprobado que compila y ejecuta sin errores, y que los resultados obtenidos son exactamente idénticos a la ejecución en Google Test puro,

podemos decir que la integración esta testeada y es correcta.

### 5.8 Iteración 7

Pasamos a la iteración 7 en la cual se procede a integrar el segundo framework en la solución, esta vez CppUnit Test.



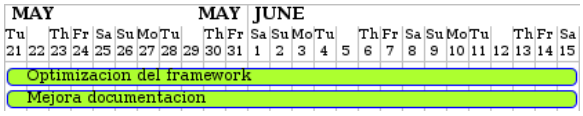

*9. Iteración 7*

Para integrarlo en la solución necesito adaptar las macros que había creado para la integración de Google Test, pero en este caso para CppUnit.

Al igual que en la fase de trabajo con los diferentes framewroks, de nuevo CppUnit me vuelve a dar problemas. En esta ocasión, la solución de este proyecto estaba preparada para frameworks en la que la declaración de los tests suites y los tests son independientes, es decir yo puedo declarar un test suite y en otro punto posterior del código, cuando sea necesario puedo declarar un test para ese test suite. En lo que había trabajado anteriormente con CppUnit anteriormente no había encontrado ese problema así que necesite encontrar la forma de poder hacerlo en este entorno. Despues de unos días investigando diversas maneras de hacerlo, encontré una forma muy sencilla con la cual pude integrar CppUnit[7] en la solución del proyecto sin demasiadas modificaciones.

Llegados a este punto, tenemos los dos frameworks integrados, aunque aun no está disponible su ejecución en ambas plataformas sin necesidad de hacer ningún cambio en el código.

### 5.9 Iteración 8



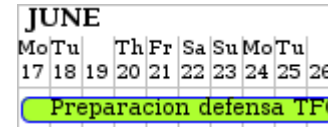

*10. Iteración 8*

Llegamos a la última iteración del desarrollo de la solución del proyecto, en esta mejoramos el framework introduciendo cambios como, ampliación de macros y la posibilidad de su ejecución en ambas plataformas sin cambios en el código, únicamente añadiendo un parámetro en la compilación.

Por otro lado, tras hablar y consultar con el tutor decidimos que la documentación, que en primera instancia iba a ser generada con Microsoft Office Word, era mejor generarla con hheramientas que fueran publicas, debido a que un documento Word es algo propio mientras que como he mencionado en los objetivos, se buscaba crear una solución libre que cualquiera pudiera utilizar e incluso trabajar y hacerla evolucionar.

Por tanto, decidí crear la documentación en dos partes. Por un lado, la documentación relacionada con el código que pasará a ser generada con doxygen, la herramienta mencionada en el punto 6.4 y por el otro lado la documentación relacionada con la preparación del entorno, la escritura de tests con la solución del proyecto, la ampliación del framework, la realización de coverage y la compilación y ejecución del test que se generará en la wiki[8] del proyecto en GitHub.

### 5.10 Iteración 9




*11. Iteración 9*

Por último, entramos en la iteración 9. Esta iteración esta dedicada a la preparación de la defensa del TFG de cara a la finalización de este.

## 6 Software y herramientas empleadas

En esta sección describiremos brevemente las herramientas y aplicaciones usadas para realizar el proyecto.

### 6.1 Microsoft Visual Studio


*12. Microsoft Visual Studio[10]*

Microsoft Visual Studio[9] es una herramienta para el desarrollo de código. En ella he trabajado sobre Google Test y el framework propio de CppUnit. Gracias a este trabajo pude entender el funcionamiento de Google Test y extraer funcionalidades que más tarde integraría en la solución de este proyecto.

### 6.2 PlantUML

PlantUML[11] es una herramienta de código abierto para crear diferentes tipos de diagrama. He utilizado esta herramienta para crear los diferentes diagramas de Gantt[12]

que he necesitado a lo largo del proyecto tanto para definir las diferentes iteraciones como para plasmarlas gráficamente en cada uno de los informes que se han tenido que ir generando. También en esta herramienta he generado el diagrama de flujo para mostrar la relación de los ficheros en la compilación y ejecución del test.

### 6.3 C++

*13. C++[14]*

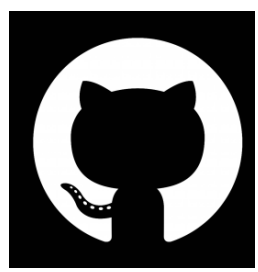
C++[13] ha sido el lenguaje elegido para generar la solución. Tanto la propia solución como la aplicación de prueba se han generado usando este lenguaje. Es un lenguaje diseñado por Bjarne Strustrup, que extiende el lenguaje C con mecanismos para poder manipular objetos, es decir, es un lenguaje orientado a objetos.

### 6.4 Doxygen


*14. Doxygen[16]*

Doxygen[15] es una herramienta estándar para la generación de documentación a partir de anotaciones en el código fuente. Es compatible con diversos lenguajes de programación y principalmente con C++. Puede generar diversos outputs de salida como LaTeX y pdf entre otros. Gracias a esta herramienta he generado la documentación relacionada con la aplicación de prueba, y más importante todavía, con la solución del framework.

### 6.5 GitHub



*15. GitHub[18]*

GitHub[17] es un repositorio online donde almacenar código. Es la plataforma líder en su ámbito y además de guardar código se puede generar la documentación deseada en sus wikis y los usuarios pueden hacer comentarios sobre el código generando una especie de foro. En esta herramienta se alojará el código del framework y la documentación generada con doxygen. Además, tendrá una wiki con más documentación sobre el framework (como escribir tests, instalación de herramientas, etc).

### 6.6 Git

*16. Git[20]*

Git[19] es un software de control de versiones. Esta herramienta se sincroniza con diferentes repositorios y permite mediante una consola subir avances en el proyecto, crear brancas de desarrollo, etiquetar versiones, etc. Esta herramienta fue creada originalmente por Linus Torvalds y es una herramienta de software libre disponible para sistemas Linux y Windows.

### 6.7 Editor de textos (Linux)

En el editor de textos de Linux, primero trabajé con la plataforma de CppUnit sobre la aplicación de prueba, debido a que con Microsoft Visual Studio tuve muchos problemas y no conseguia compilar la aplicación, así que decidí hacerlo directamente en Linux, donde iba a desarrollar la solución final del framework.

### 6.8 Bitbucket

Bitbucktet[21] al igual que GitHub es un repositorio online donde almacenar código. Aunque es una buena herramienta, esta más enfocada a repositorios privados y no dispone de la posibilidad de crear documentación asociada al proyecto, principales razones por las que se ha decidido alojar el código final en GitHub.

## 7 RESULTADOS

A continuación, comentaremos los resultados más destacables extraidos de la realización del proyecto.

### 7.1 Mejoras frente a otros frameworks

**Portabilidad y reusabilidad:** A diferencia de otros frameworks, entre ellos los dos integrados en esta solución, Google Test y CppUnnit Test, en que estan realizados para ejecutarse dentro de su entrono y plataforma, este framework permite escribir test que luego pueden ser ejecutados en plataformas diferentes, por ahora en las dos integradas, pero puede ser ampliado a otras plataformas si es necesario.

**Facil ampliación y uso:** El código de este framework esta bastante modularizado, teniendo las diferentes partes del código separadas en diferentes archivos por lo que cada uno de sus modulos es independiente convirtiéndolo en un código sencillo de entender y por tanto sencillo de modificar ya que permite tener claro que hay que hacer en cada lugar.

**Más intuitivo:** Este framework tiene expandidas las macros para hacer su escritura mas intuitiva. Por ejemplo, si necesitas comparar un numero mayor a otro, un string igual a otro, dos números iguales, etc, cada una tiene su propia macro en la que su nombre indica que es lo que va a hacer esa macro además de simplificarlas.

### 7.2 Test

El test sobre el framework desarrollado en este proyecto ha consistido básicamente en la comprobación de que, ejecutando el test en las diferentes plataformas, obteníamos los mismos resultados que haciendolo con el código de test puro en cada una de ellas. Esto se debe principalmente debido a que realizar test sobre el test no tiene mucho sentido porque entraríamos en un bucle infinito de test del cual nunca saldríamos.

## 8 Conclusiones

Para terminar, decir que el proyecto ha cumplido con los objetivos que se habían marcado al inicio de este.

Despues de investigar los frameworks más usados, y ver que gran parte de la comunidad coincidia, podemos decir que estos son: Google Test, CppUnit y Boost Test.

Tras trabajar con dos de ellos y ver si era posible una solución común, he podido concluir que si, debido a que ambos tienes una estructura similar (declaración del test suite y posterior adición de tests). Así pues, con la conclusión positiva se ha generado una solución libre y compatible con entornos libres como GNU G++.

Cabe mencionar que, en mi opinión, una solución como esta puede ser muy beneficiosa en el mundo de la informática ya que, aunque en un primer momento pueda ocupar un tiempo utilizar esta solución en caso de que el entorno de test que quieras utilizar no este integrado y debas integrarlo, en un futuro te puede aportar un ahorro de tiempo muy valioso en la creación de test debido a la portabilidad que aporta este framework.

Finalmente, el objetivo secundario, no se ha podido realizar debido a que no se ha tenido el tiempo suficiente cómo para trabajarlo.

**Posibles mejoras:** Además de los objetivos futuros mencionados al principio del informe, actualmente esta solución no cubre el 100% de las posibilidades de test de las dos plataformas integradas en la solución, aunque la mayoría de las funcionalidades restantes son derivaciones de las ya implementadas, pero añadiendo mensajes de salida o funcionalidades para la comprobación del lanzamiento de excepciones.

### Objetivos futuros

Cómo planes de futuro para este proyecto, se quiere realizar una ampliación del framework que ha resultado. Para ello, tenemos dos posibles vertientes.

La primera consiste en ampliar las macros disponibles para las dos plataformas ya integradas (Google Test, CppUNit). Hay diversas funcionalidades integradas con las que se puede trabajar y realizar Unit Tests, pero es posible que aparezcan nuevas funcionalidades, o que no haya identificado algunas de ellas. Integrando estas funcionalidades mencionadas conseguiríamos la primera vertiente de ampliación.

La segunda vertiente consiste en la integración de nuevas plataformas de test no integradas en esta solución. Esto consistiría en analizar la nueva plataforma y adaptar las funcionalidades de ambas para que se correspondan.

Este objetivo de futuro puede ser realizado tanto por mi mismo cómo por integrantes de la comunidad informática ya que es un proyecto libre en el que cualquiera puede aportar su grano de arena para hacerlo crecer.

## Agradecimientos

Agradecer en primer lugar a Lluis Gesa Boté por todo el soporte, apoyo y todos los consejos que me ha preporcionado durante la realización del proyecto, des del primer dia hasta el último.

Agradecer también a todos los amigos y familiares que me han apoyado en esos momentos con los animos bajos en los que alguna parte del desarrollo no conseguia sacarla adelante.

¡Gracias!

## Bibliografia

[1] Google | Google Test [online] https://github.com/google/googletest

[2] CppUnit | CppUnit [online] http://cppunit.sourceforge.net/doc/cvs/index.html

[3] Boost C++ Libraries | Boost Test [online] https://www.boost.org/doc/libs/1_70_0/libs/test/doc/html/index.html

[4] Proyectos Agiles | Desarrollo iterativo e incremental [Online] https://proyectosagiles.org/desarrollo-iterativo-incremental/

[5] Modelo incremental. [Ilustración]. Recuperado de http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro17/12_etapas.html

[6] SNL | Metodologia Kanban [online] https://www.snl19.es/5-claves-gestion-proyectos-metodologia-kanban/

[7] IBM | Incorporating new tests [online] https://www.ibm.com/developerworks/aix/library/au-ctools2_cppunit/index.html

[8] Sistema Unit Test | Wiki [online] https://github.com/FranciscoMagdaleno/SistemaUnitT

est/wiki
[9] Wikipedia | Microsoft Visual Studio [online] https://es.wikipedia.org/wiki/Microsoft_Visual_Studio
[10] Micrososft Visual Studio. [Ilustración]. Recuperado de http://codigoresuelto.com/
[11] PlantUML | PlantUML de un vistazo [online] http://plantuml.com/es/
[12] Sinnaps | Diagrama de Gantt [online] https://www.sinnaps.com/blog-gestion-proyectos/diagrama-gantt-sirve
[13] Wikipedia | C++ [online] https://es.wikipedia.org/wiki/C%2B%2B
[14] C++. [Ilustración]. Recuperado de https://es.wikipedia.org/wiki/C%2B%2B
[15] Doxygen | Doxygen [online] http://www.doxygen.nl/
[16] Doxygen. [Ilustración]. Recuperado de http://migracion.ucr.ac.cr/programas/doxygen/
[17] Wikipedia | GitHub [online] https://es.wikipedia.org/wiki/GitHub
[18] Github. [Ilustración]. Recuperado de https://github.com/github
[19] Wikipedia | Git [online] https://es.wikipedia.org/wiki/Git
[20] Git. [Ilustración]. Recuperado de https://medium.com/javascript-comunidad/iniciando-con-git-2cf89439c862
[21] Atlassian | BitBucket [online] https://bitbucket.org/

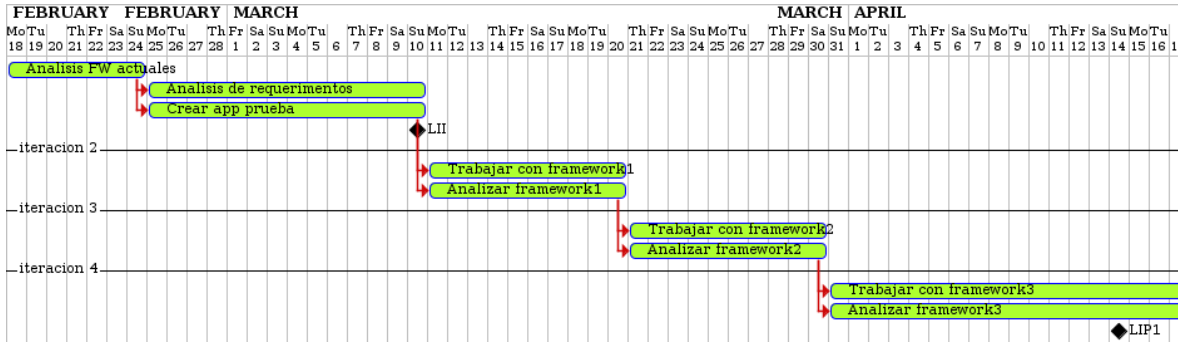
# APENDICE

En este apéndice podréis encontrar información extra del proyecto.

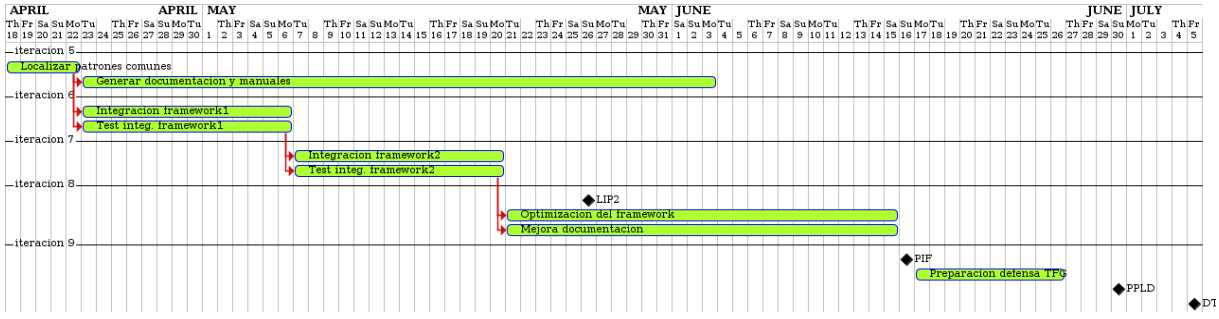
## A1. INDEX DE IMÁGENES



## A2. DIAGRAMA DE GANTT COMPLETO




*17. Diagrama de Gantt parte 1*




*18. Diagrama de Gantt parte 2*

## A3. CAPTURAS DE PANTALLA


*19. Wiki del proyecto*

### A3.1 Wiki

En esta imagen podemos observar la wiki del proyecto donde podemos consultar diversa información como la instalación de entornos, la compilación y ejecución del framework, …

```
francisco@francisco-VirtualBox:~/Escritorio/TFG/Test$ g++ -DCPPUNIT_TEST_ENABLED
 AllTests.cpp test_calculadora.cpp ../Codi/calculadora.cpp -o cppUnit -lcppunit
francisco@francisco-VirtualBox:~/Escritorio/TFG/Test$ ./cppUnit
(anonymous namespace)::ctest_calculadoratest_cal_run_tests::runtest_cal_run_test
s : OK
(anonymous namespace)::ctest_calculadoratest_suma::runtest_suma : OK
OK (2)
francisco@francisco-VirtualBox:~/Escritorio/TFG/Test$
```

*20. Ejecucion sobre CppUnit*

### A3.2 Ejecución en CppUnit

En esta captura podemos observar el comando para compilar en CppUnit y su posterior ejecución con los resultados

```
francisco@francisco-VirtualBox:~/Escritorio/TFG/Test$ g++ -DGOOGLE_TEST_ENABLED
AllTests.cpp test_calculadora.cpp ../Codi/calculadora.cpp -o gtest /usr/local/li
b/libgtest.a -lpthread
francisco@francisco-VirtualBox:~/Escritorio/TFG/Test$ ./gtest
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from ctest_calculadora
[ RUN      ] ctest_calculadora.test_cal_run_tests
[       OK ] ctest_calculadora.test_cal_run_tests (0 ms)
[ RUN      ] ctest_calculadora.test_suma
[       OK ] ctest_calculadora.test_suma (0 ms)
[----------] 2 tests from ctest_calculadora (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 2 tests.
francisco@francisco-VirtualBox:~/Escritorio/TFG/Test$
```

*21. Ejecución sobre Google Test*

### A3.3 Ejecución sobre Google Test

Aquí podemos observar, igual que en la anterior, los resultados de compilación y ejecución, esta vez sobre Google Test