
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN**

# SIMFORPAS

## Simulador para RPAS

**Realizado por**

**MANUEL MATEOS GUTIÉRREZ**
**32076954-G**

**Dirigido por**

**IRENE ALEJO TEISSIÈRE**

**PABLO TRINIDAD MARTÍN-ARROYO**

**Departamento**

**LENGUAJES Y SISTEMAS INFORMÁTICOS**

**Sevilla, Mayo de 2014**

# Agradecimientos

Agradecimientos

# Índice general


**I Introducción** **1**

**1. Introducción** **3**
1.1. Motivación . . . . . 4
1.2. Objetivos del proyecto . . . . . 4
1.2.1. Objetivos orientados a la metodología . . . . . 5
1.2.2. Objetivos orientados a la técnica . . . . . 5
1.2.3. Objetivos personales . . . . . 6
1.3. Estructura del documento . . . . . 6

**II Conceptos básicos** **9**

**2. Metodologías usadas** **11**
2.1. Metodologías ágiles . . . . . 11
2.1.1. Manifiesto ágil . . . . . 12
2.1.2. Desarrollo iterativo incremental con Scrum . . . . . 13
2.1.3. Test Driven Development . . . . . 16
2.1.4. Pair programming . . . . . 17

# Índice de figuras

1.1. Soporte aéreo en el control de incendios. . . . . 3

2.1. Manifiesto ágil. . . . . 12
2.2. Ciclo metodología Scrum. . . . . 15
2.3. Ciclo TDD. . . . . 16

# Parte I

# Introducción

# Capítulo 1. Introducción

AUNQUE el concepto de aviones no tripulados o UAV's (Unmanned Aerial Vehicles) es bastante antiguo, puesto que se ha hecho uso de ellos desde la primera guerra mundial, cada día oímos más hablar sobre ellos en los medios de comunicación, esto se debe al gran crecimiento que está sufriendo el sector de la aeronáutica en torno a estos dispositivos, tanto para uso militar, como los famosos "drones"de Estados Unidos, como civil.

Su uso es amplio y variado, desde rodaje de planos aéreos en películas de cine hasta control de incendios, control de costas, recogida de información, ayuda en operaciones de rescate, control de multitudes...


Figura 1.1: Soporte aéreo en el control de incendios.

A finales del siglo XX fue cuando los UAV's empiezan a operar con todas las características de autonomía. Esto nos provee de muchas ventajas, por ejemplo, presencia en lugares de difícil acceso sin necesidad de llevar al terreno a un piloto de UAV, reducción del riesgo humano en determinadas situaciones, disminución del la incursión humana sobre parques naturales y zonas protegidas... . Poco a poco los UAV's tienden a prescindir de la presencia de un piloto que tenga la obligación de estar visualizando el avión y a implementar sistemas de control remoto mediante estaciones de control de tierra o GCS's (Ground Control Stations) y de vuelo automatizado, lo que nos llevará a no depender del factor humano.

Las GCS son controladas por operadores expertos en estos dispositivos que se encargan de diseñar e implementar las misiones que realizarán los aviones, así como llevar el control del curso de la misma, conocer las características de la aeronave, deben saber interpretar

los indicadores de telemetría, estar familiarizados con el protocolo de comunicación y saber reaccionar ante posibles fallos durante la misión para salvaguardar en todo momento la seguridad tanto del vehículo aéreo como del entorno en el que se mueve.

Estos operadores requieren de una formación en profundidad y fiable ya que tienen la responsabilidad sobre las acciones que realice la aeronave, por ello se debe exigir un entrenamiento concienzudo. Si este entrenamiento es realizado con dispositivos reales corremos el riesgo de que frente a cualquier fallo, error humano o de carácter informático, haya una pérdida en algún componente del sistema, ya sea que se estrelle la aeronave, que dañe alguna estructura o a alguna persona, lo que resultaría en una importante pérdida económica y/o humana.

El proyecto SIMFORPAS pretende dar una solución a este problema presentando un entorno de simulación de vuelo de UAV's para operadores de GCS en formación, que proveerá de un contexto de vuelo seguro e idéntico a una situación real de control de misión de un UAV.

## 1.1. Motivación

Según un estudio publicado por el medio online *Update Defense* y realizado por la firma de investigación de mercados *ICD Research* durante la próxima década el sector de la aviación no tripulada tendrá un aumento anual del 4,08 % lo que supondrá que en 2021 alcance alrededor de los 10.500 millones de dólares. El gran incremento de la demanda se traducirá en una mayor necesidad de infraestructuras y tecnologías en torno a estos dispositivos.

En vistas de estas espectativas resulta interesante implicarse de una forma activa en un mercado en auge que supondrá la aceptación de un gran número de nuevas tecnologías y traerá nuevos desafíos en cuanto a investigación y desarrollo.

Uno de los requisitos que tendrá esta etapa será pa de disponer de personal cualificado para la manipulación de los dispositivos de pilotage remotos de aeronaves no tripuladas, el proyecto SIMFORPAS pretende ocupar ese hueco proveyendo de un entorno seguro y fiable que permita conceder una certificación avanzada a operadores de GCS de forma que se aseguren los conocimientos técnicos necesarios en una situación real de pilotage.

En este proyecto propondremos una solución usando una serie de tecnologías que nos proveerán de las herramientas necesarias para crear el sistema necesario para la consecución de nuestro objetivo.

## 1.2. Objetivos del proyecto

El objetivo de este proyecto será el de crear una plataforma de simulación para la formación y entrenamiento de pilotos de RPAS (Remotely Piloted Aircraft System) ligeros que se comporte exactamente como lo haría el avión real. Que el sistema permita

hacer uso de una GCS homologada para el manejo de UAV's usando un protocolo de comunicaciones para aviones no tripulados de menos de 25 Kg y usando un modelo de avión real.

También se requerirá de una herramienta que permita a un instructor ser capaz de controlar la simulación permitiéndole manejar su curso e introducir errores en el sistema. La simulación se deberá hacer en tiempo real.

Para garantizar la correcta consecución de los objetivos generales del proyecto se utilizará la herramienta de organización SCRUM junto a otras metodologías de desarrollo ágil.

### 1.2.1. OBJETIVOS ORIENTADOS A LA METODOLOGÍA

Los objetivos que se tienen en mente al realizar esta aplicación con respecto a las metodologías usadas son los siguientes:

- ⋆ Aplicar el marco de trabajo Scrum, usando para ello los conocimientos adquiridos al trabajar en empresas que utilizan dicha metodología y cursos. También se dispone del apoyo bibliográfico de libros como *Agile Samurai* y *Agile Software Development with Scrum*
- ⋆ Aplicar metodologías de programación en pareja para agilizar el desarrollo y evitar errores en el código.
- ⋆ Aplicar los principios S.O.L.I.D. como base de un código robusto, limpio y sujeto a cambios.
- ⋆ Aplicar metodologías de eXtreme Programming para asegurar que el código acepte cambios de manera sencilla e intuitiva.
- ⋆ Comprobar los beneficios colaterales a la realización de estas prácticas como, por ejemplo, la facilidad de añadir nuevas tareas durante el proceso de desarrollo.

### 1.2.2. OBJETIVOS ORIENTADOS A LA TÉCNICA

Los objetivos que se han querido validar al realizar esta aplicación son los siguientes:

- ⋆ Aprender y utilizar tecnologías que garanticen una comunicación y procesado de datos en tiempo real como pueden ser C++ y DDS.
- ⋆ Aprender y utilizar para el puesto de instructor herramientas que ayuden al desarrollo de una plataforma web para el control del simulador como Maven, Struts2, Spring4 e Hibernate4.
- ⋆ Aprender y utilizar entornos de testeo de código como Google test, Google mock, Junit y Jmock para asegurar que el código funciona en todas las fases de desarrollo y modificación del software.

⋆ Aplicar correctamente cada una de las metodologías estudiadas y sacar conclusiones de su uso.

### 1.2.3. OBJETIVOS PERSONALES

Se ha querido asegurar que se cumplen los siguientes objetivos a lo largo del desarrollo de la aplicación:

⋆ Adaptación: Adecuarse a las exigencias de estas nuevas metodologías. Los desarrolladores tienen la motivación de aprender nuevas técnicas que mejoren la calidad del software.

⋆ Confianza: Conseguir conocimientos que me permitan en un futuro abatir exitosamente un proyecto software.

⋆ Experiencia: Adquirir aptitudes para solucionar problemas y añadir valor a un grupo de trabajo en un entorno laboral.

⋆ Conocimientos Técnicos: Trabajar y sintetizar nuevas tecnologías que me sirvan en el futuro para completarme como profesional en mi campo.

Se tendrán en cuenta estos objetivos durante la realización del proyecto y se comprobará si han sido realizados. Esto se verá con detenimiento en los apartados de conclusiones, véase la parte V del presente documento.

## 1.3. ESTRUCTURA DEL DOCUMENTO

Éste documento se estructura en las siguientes partes:

**PREFACIO:** En éste capítulo se introduce el proyecto creando el contexto de su implementación, explicando en qué consiste, la motivación que nos ha llevado a desarrollarlo y los objetivos que se quieren cumplir en el mismo, así como éste mismo apartado de estructura del proyecto en el que explicamos qué vamos a encontrarnos en ésta memoria y cómo está distribuida y un índice de contenidos y de figuras.

**CONCEPTOS BÁSICOS:** Introducimos las metodologías que hemos usado durante el desarrollo del proyecto así como los conceptos básicos necesarios para comprender todo el documento, una enumeración de tecnologías usadas y un glosario de terminología, también se explicará el método de desarrollo iterativo e incremental que hemos llevado a cabo y en el que se basa la documentación del proyecto.

**SISTEMA A DESARROLLAR:** Aquí se enumerarán cada una de las etapas de desarrollo que ha ido sufriendo el proyecto SIMFORPAS desarrollando la planificación para cada iteración y cada problema que ha ido surgiendo durante el mismo, también se aportará el diagrama de Burndown para monitorizar en todo momento el estado del proyecto.

**CONCLUSIONES:** Realizaremos una retrospectiva final del proyecto analizando su estado final, la consecución de los objetivos, los cambios con respecto a la planificación inicial que se han realizado y las posibles mejoras y futuro del proyecto SIMFORPAS.

**APENDICES:** Para finalizar añadiremos un apéndice de definiciones, un manual de usuario y la bibliografía usada durante el desarrollo del proyecto y la memoria.

# Parte II

# Conceptos básicos

# Capítulo 2. Metodologías usadas

En éste capítulo empezaremos haciendo una introducción a las metodologías ágiles, explicando su filosofía y el por qué de su existencia así como una serie de técnicas para implementar este tipo de metodologías a nuestro proyecto software y qué beneficio nos aporta.

El proyecto fue desarrollado haciendo uso del sistema SCRUM de desarrollo iterativo, se explicará en qué consiste éste método y como ha sido aplicado a SIMFORPAS.

## 2.1. Metodologías ágiles

El proceso normal afianzado hasta ahora en el desarrollo software sigue unas pautas de rigidez que evita que el producto esté sometido a cambios ya que cuanto más avanzado está el desarrollo del proyecto más difícil y costoso resulta la introducción de modificaciones, para ello se definen unos requisitos que debe cumplir el producto final y antes de empezar el proyecto se decide las tecnologías a usar y la planificación del desarrollo, el cliente no toma parte en el proceso de implementación sino que cuando llega la fecha indicada para la finalización se le presenta y se evalúa si se ha conseguido el resultado que él esperaba.

Como pueden imaginar en la mayoría de los casos debido al desconocimiento real del problema no se definen correctamente los requisitos o las tecnologías usadas y surgían problemas imprevistos en la planificación que retrasan la fecha de entrega o acortan el tiempo de desarrollo obligando al equipo a dedicar más horas repercutiendo todo esto negativamente en el resultado final.

En otros casos la entrega se hace a tiempo pero debido a la ausencia del cliente durante el proceso de desarrollo el producto final no responde a lo que él imaginaba que se iba a desarrollar causando descontento por parte de nuestro cliente y afectando a futuros contratos que podamos hacer con él mismo.

Para hacer frente a esta serie de problemas en torno al desarrollo software nacen las ”Metodologías Ágiles”, En 2001 un grupo de desarrolladores se reúne en Utah para discutir los *métodos de peso ligero* de desarrollo software y publicaron el *Manifiesto ágil*, un documento que resume la filosofía ágil y establece cuatro valores y doce principios.

### 2.1.1. MANIFIESTO ÁGIL

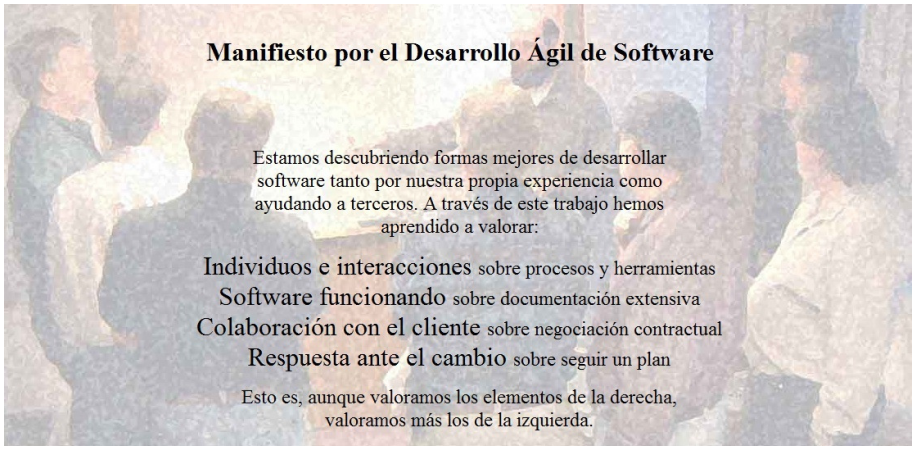


Figura 2.1: Manifiesto ágil.

#### VALORES:

- ⋆ **Valorar más a los individuos y su interacción que a los procesos y las herramientas:** Este es posiblemente el principio más importante del manifiesto. Por supuesto que los procesos ayudan al trabajo. Son una guía de operación. Las herramientas mejoran la eficiencia, pero sin personas con conocimiento técnico y actitud adecuada, no producen resultados.
- ⋆ **Valorar más el software que funciona que la documentación exhaustiva:** La documentación siempre será una medida importante pero no como guía para entender un código sino como complemento de un código claro y autoexplicativo. Al final lo que se debe valorar es un código ordenado y que funciona, que le da valor a un proyecto, por encima de una documentación que aporta datos y no información.
- ⋆ **Valorar más la colaboración con el cliente que la negociación contractual:** Las prácticas ágiles están especialmente indicadas para productos difíciles de definir con detalle en el principio, o que si se definieran así tendrían al final menos valor que si se van enriqueciendo con retro-información continua durante el desarrollo. También para los casos en los que los requisitos van a ser muy inestables por la velocidad del entorno de negocio. En el desarrollo ágil el cliente es un miembro más del equipo, que se integra y colabora en el grupo de trabajo. Los modelos de contrato por obra no encajan.
- ⋆ **Valorar más la respuesta al cambio que el seguimiento de un plan:** Para un modelo de desarrollo que surge de entornos inestables, que tienen como factor inherente el cambio y la evolución rápida y continua, resulta mucho más valiosa la capacidad de respuesta que la de seguimiento y aseguramiento de planes pre-establecidos. Los principales valores de la gestión ágil son la anticipación y la adaptación; diferentes a los de la gestión de proyectos ortodoxa: planificación y control para evitar desviaciones sobre el plan.

### PRINCIPIOS:

**1**.- La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporten valor.

**2**.- Dar la bienvenida a los cambios de requisitos. Se capturan los cambios para que el cliente tenga una ventaja competitiva.

**3**.- Liberar software que funcione frecuentemente, desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.

**4**.- Los miembros del negocio y los desarrolladores deben trabajar juntos diariamente a lo largo del proyecto.

**5**.- Construir el proyecto en torno a individuos motivados. Darles el entorno y apoyo que necesiten y confiar a en ellos para conseguir finalizar el trabajo.

**6**.- El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.

**7**.- El software que funciona es la principal medida de progreso.

**8**.- Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.

**9**.- La atención continua a la calidad técnica y al buen diseño mejora la agilidad.

**10**.- La simplicidad es esencial.

**11**.- Las mejores arquitecturas, requisitos y diseños surgen de los equipos que se organizan ellos mismos.

**12**.- En intervalos regulares, el equipo debe reflexionar sobre cómo ser más efectivo y, según estas reflexiones, ajustar su comportamiento.

## 2.1.2. DESARROLLO ITERATIVO INCREMENTAL CON SCRUM

### INTRODUCCIÓN

PARA abordar la realización del proyecto SIMFORPAS se hará uso del modelo organizativo de trabajo Scrum. Una de las características de éste modelo es la búsqueda de una serie de beneficios, como por ejemplo, la capacidad de aceptación de nuevos cambios durante el desarrollo ya sean requeridos por el cliente como por el mercado, esto nos asegura que el cliente al finalizar el proyecto va a tener el producto que satisface a sus necesidades ya que de otro modo los requisitos pueden haber cambiado desde la definición inicial.

El equipo de trabajo se auto asignará las tareas a realizar, esto provoca que cada integrante se mantenga motivado ya que él mismo se ha puesto su objetivo. El desarrollo iterativo exige tener una versión funcional o una serie de resultados presentables al finalizar cada etapa del desarrollo, esto se traduce en una mayor calidad del software.

Utilizando herramientas como la gráfica de burn down es posible observar la velocidad que está llevando el equipo de desarrollo, esto es útil para detectar posibles problemas de rendimiento que haya que solucionar entre todo el equipo o una reorganización de la planificación así como para poder estimar el tiempo de duración del proyecto.

### ROLES

- ⋆ **Product owner:** El Product Owner representa la voz del cliente. Se asegura de que el equipo Scrum trabaje de forma adecuada desde la perspectiva del negocio. El Product Owner escribe historias de usuario, las prioriza, y las coloca en el Product Backlog.
- ⋆ **ScrumMaster:** El Scrum es facilitado por un ScrumMaster, cuyo trabajo primario es eliminar los obstáculos que impiden que el equipo alcance el objetivo del sprint. El ScrumMaster no es el líder del equipo (porque ellos se auto-organizan), sino que actúa como una protección entre el equipo y cualquier influencia que le distraiga. El ScrumMaster se asegura de que el proceso Scrum se utiliza como es debido. El ScrumMaster es el que hace que las reglas se cumplan.
- ⋆ **Equipo de desarrollo:** El equipo tiene la responsabilidad de entregar el producto. Un pequeño equipo de 3 a 9 personas con las habilidades transversales necesarias para realizar el trabajo (análisis, diseño, desarrollo, pruebas, documentación, etc).

Existen otros roles auxiliares como pueden ser proveedores, clientes, vendedores... sólo participarán directamente durante las revisiones de sprint.

### DESARROLLO DEL PROYECTO CON SCRUM

Al principio se tiene una reunión con el cliente donde se recoge el objetivo del proyecto, los requisitos y las tareas que se llevarán a cabo para realizarlos, todo ello mediante historias de usuario en las que se asocia el rol a la necesidad del proyecto como se puede observar en el siguiente ejemplo: *Como desarrollador quiero un módulo central capaz de cambiar y monitorizar el estado del modelo*, de esta forma se deciden las tareas a realizar. Luego el equipo y el cliente discuten la prioridad en las tareas hasta llegar a un consenso de qué es más importante desarrollar primero y qué dejar para más adelante, de esta forma nos aseguramos de ir cumpliendo las necesidades más importantes para poder tener cuanto antes una versión funcional del proyecto. También se valorarán según la dificultad de cada una de ellas, facilitando de esta forma la elección de qué se realizará antes, las acciones que sean esenciales y fáciles se harán primero, y las difíciles y poco necesarias se dejarán para el final, el tiempo de realización de cada tarea se hará en función a la dificultad de la misma.

Una vez definidas las historias de usuario se define el tamaño de los *sprints*, normalmente un sprint es un espacio de tiempo de entre una y cuatro semanas en las que se desarrollarán determinadas historias de usuario. Cuando se define el primer sprint se colocan en una pizarra las historias de usuario, para cada historia se definirán unos test de aceptación que asegurarán una vez cumplidos que la tarea está finalizada y se colocarán pequeñas

sub-tareas necesarias para la realización de la historia de usuario. La pizarra tendrá varios *pools* que indicarán las sub-tareas a realizar, las que están en proceso y las que ya se han realizado. Éstas sub-tareas se irán cambiando de posición según sea su estado. La morfología de la pizarra de Scrum se muestra en la siguiente figura.

Durante el sprint se hará una reunión diaria entre el equipo y el ScrumMaster en la que se hablará del estado del proyecto, qué se realizó el día anterior, qué se realizará en ese día y qué problemas han surgido para buscar entre todos soluciones y que ningún miembro del equipo se quede estancado en una tarea.

Una vez finalizado el sprint se organiza una reunión de retrospectiva, a la que acudirá el equipo, el ScrumMaster y el product owner en la que se presentará el estado del proyecto, qué es lo que se ha llevado a cabo durante el sprint, qué problemas han surgido, que se podría modificar/mejorar. El cliente dará el visto bueno y hará las peticiones que vea necesarias, se hará la gráfica de burn down para documentar el estado de esa fase del proyecto y se organizarán las tareas para el siguiente sprint.

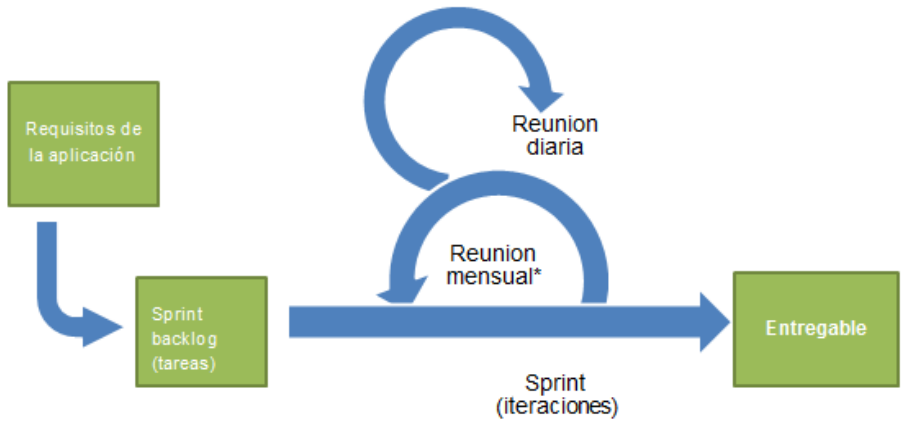


Figura 2.2: Ciclo metodología Scrum.

Esta forma de trabajo se repetirá hasta la finalización del proyecto, nos asegurará que el cliente está implicado en el desarrollo y que conocerá de antemano el producto que va a comprar y podrá interceder en su morfología. Con este método el equipo de desarrollo trabaja de una forma más relajada evitando la acumulación de trabajo a última hora y los estancamientos ya que entre el equipo debe fluir la comunicación y el problema que tenga uno en la realización de su parte se convierte en problema de todos. El cliente podrá añadir cambios o complementos al proyecto a sabiendas de que esos cambios vendrán con el sacrificio de otras historias de usuario programadas o de un incremento del tiempo de desarrollo y del coste del proyecto.

### 2.1.3. TEST DRIVEN DEVELOPMENT

EL desarrollo guiado por pruebas o TDD por sus siglas en inglés consiste en una práctica de programación que implica a su vez otras dos prácticas: Escribir las pruebas antes que el código y refactorizar. Una vez habiendo definido la funcionalidad de la parte del código que vamos a escribir hacemos un test que pruebe esa funcionalidad y una vez definido éste test y fallando nos disponemos a codificar la solución que lo resuelva, de esta forma nos aseguramos de que no perdemos ninguna función al programar el código. Si se hace al revés el test se ve afectado por la forma que tiene el código y tendemos a probar lo que sabemos que va a ocurrir dejándonos muchos casos sin resolver que pueden afectar más tarde al correcto funcionamiento de nuestro programa. Con esta técnica también nos aseguramos que en el momento en que se realice un cambio en el programa nada deja de funcionar, puesto que en todo momento el código debe pasar los test asegurando que no se pierde ninguna funcionalidad debida al nuevo cambio.

Una vez se ha escrito el test, se ha comprobado que fallaba y se ha resuelto viene la hora de refactorizar el código, como no sabemos a priori cómo va a ser el código final debemos probablemente el código que hemos escrito para pasar ese test sea memorable, por eso tenemos que estudiar la forma correcta de escribir esa parte del código, esto se hace mediante una refactorización. Una vez realizada ésta refactorización se vuelven a pasar los test, si no pasan habría que repasar el código para que se solucione el error y luego volver a repasar la refactorización.

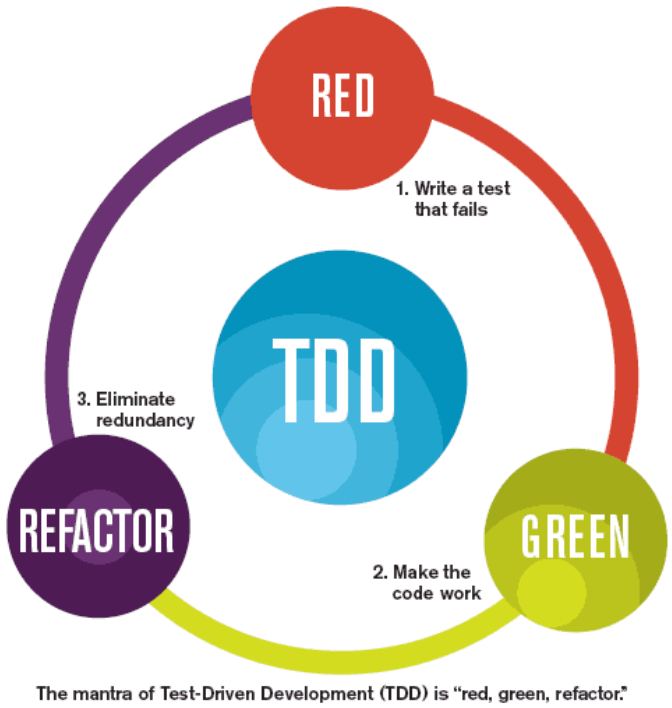


Figura 2.3: Ciclo TDD.

Mediante ésta técnica nos aseguramos un código limpio, bien estructurado y libre de errores. Otra funcionalidad de los test es explicar de qué manera se usa el código que estamos programando, ya que para probar nuestros métodos y clases debemos hacer uso de ellas, y este uso queda reflejado en el test.

### 2.1.4. PAIR PROGRAMMING

A la hora de programar es muy común perder mucho tiempo con errores al codificar así como en tomar decisiones correctas sobre qué forma darle al código, una técnica que evita estas situaciones es la programación por parejas, consiste en unir a dos desarrolladores para que programen juntos en el mismo puesto de trabajo, de forma que mientras uno programa el otro vigila que no tenga errores. También deciden entre los dos cómo hacer las cosas de una forma objetiva, siempre es bueno tener una segunda opinión y discutir cuál es la mejor solución a un problema.

A priori puede parecer que éste método hace que dos personas estén haciendo el trabajo de una, pero a la larga esto acelera el tiempo de desarrollo. También es muy útil a la hora de transmitir conocimientos a una nueva incorporación al equipo o para enseñar a programadores junior.

Cada cierto tiempo se pueden intercambiar los papeles lo que les permitirá a las dos partes coger soltura y ver el código con perspectiva de forma que puedan abstraerse y tener una visión global. Ésta técnica puede combinarse con la programación guiada por tests de forma que uno de los dos escribe el test y el otro tiene que escribir el código que lo resuelve para que el otro tenga la obligación de pensar de qué forma podría fallar y así tener una mayor cobertura frente a fallos.

Figura 2.4: Ciclo TDD.