4. MÉTODO DE TRABAJO

En el presente capítulo se explicará y analizará la metodología de gestión escogida para el desarrollo del TFG. Primero se hablará sobre las metodologías ágiles y a continuación se describe la metodología eXtreme Programming (XP), que es la que se ha usado para el desarrollo de software. Por último, se explica como se ha utilizado XP en el proyecto.

4.1 Metodologías ágiles

Las metodologías ágiles describen un conjunto de valores y principios para el desarrollo de software bajo los cuales los requisitos y las soluciones evolucionan a través del esfuerzo colaborativo de equipos organizados multifuncionales.

Usan una planificación adaptativa, el desarrollo evolutivo, la entrega temprana y la mejora continua fomentando una respuesta rápida y flexible al cambio.

Fue desarrollado como evolución de las clásicas metodologías de gestión. En 2001 varios desarrolladores de software se reunieron en Utah para analizar estos métodos ágiles, publicando así el Manifiesto para el desarrollo de software ágil. Este documento está dividido en doce principios y cuatro valores generando así la filosofía de las metodologías ágiles.

Entre los distintos métodos o técnicas basadas en metodología de desarrollo ágil, los más populares son:

- eXtreme Programming. Es el método utilizado en el presente TFG y se detalla más adelante.
- Scrum. Es una metodología ágil que proporciona un marco de trabajo para la gestión de proyectos. En la actualidad es una de las más populares. Tiene como objetivo obtener resultados tempranos adaptándose a cambios en las necesidades de los clientes. Entre sus principios más característicos se encuentran el desarrollo software por medio de iteraciones incrementales y las reuniones a lo largo del proyecto.
- Dynamic Systems Development Method (DSDM). Este método se apoya en la continua interacción con el usuario. Esto da lugar a un sistema mucho más adaptable a los requisitos cambiantes del usuario y a las necesidades de la empresa.

- Agile Unified Process (AUP). Muestra un enfoque simple para el desarrollo de aplicaciones comerciales por medio de técnicas y conceptos ágiles. Estas técnicas incluyen el desarrollo basado en pruebas, el modelado ágil, gestión ágil de cambios y refactorización de bases de datos para mejorar la productividad.

4.1.1 Diferencia entre metodologías ágiles y tradicionales

Generalmente las metodologías tradicionales tienen como prioridad el uso exhaustivo de documentación y tienen como meta cumplir con la planificación del proyecto dejando de lado la capacidad de respuesta a los cambios, así como la importancia de la confianza en las habilidades del equipo y las relaciones con el cliente.

En Fig 4.1 se puede observar una comparación de los ciclos de desarrollo software entre la metodología tradicional y la ágil. Como principal diferencia, se puede observar la existencia de iteraciones en la metodología ágil, ocupando una gran parte del ciclo de desarrollo.

Habiendo visto la comparativa en los ciclos de desarrollo, en el Cuadro 4.1 se muestran las diferencias respecto al contexto del equipo y organización.

4.1.2 Manifiesto ágil: Valores

El Manifiesto Ágil propuso un conjunto de valores y principios para todas las metodologías ágiles de aquel momento como contraposición a las clásicas metodologías de gestión desarrolladas hasta ese momento. Estos valores se disponen como una contraposición. Estos valores son los siguientes:

- Individuos e interacciones sobre procesos y herramientas. Se refiere al hecho de que lo más importante en el desarrollo de software son las personas y las interacciones entre ellas, por lo que es prioritario su bienestar individual y común. Este valor es clave para mejorar la productividad en equipo.
- Software de trabajo sobre documentación extensiva. Este valor define la tendencia a documentar de manera excesiva los desarrollos software de manera que se emplea más tiempo en la documentación que en el propio desarrollo de software.
- Colaboración del cliente sobre negociación contractual. Se refiere al hecho de crear contratos de requisitos menos cerrados dando lugar a una interacción constante entre el cliente y el equipo de desarrollo.

- Respondiendo ante el cambio sobre seguir un plan. Este valor define la inviabilidad de una planificación al uso debido a que las metodologías ágiles necesitan cambios constantes en los requisitos durante el desarrollo de software.

Metodología Ágil	Metodología Tradicional
Pocos Artefactos. El modelado es prescindible, modelos desechables.	Más Artefactos. El modelado es esencial, mantenimiento de modelos
Pocos Roles, más genéricos y flexibles	Más Roles, más específicos
No existe un contrato tradicional, debe ser bastante flexible	Existe un contrato prefijado
Cliente es parte del equipo de desarrollo (además in-situ)	El cliente interactúa con el equipo de desarrollo mediante reuniones
Orientada a proyectos pequeños. Corta duración (o entregas frecuentes), equipos pequeños (< 10 integrantes) y trabajando en el mismo sitio	Aplicables a proyectos de cualquier tamaño, pero suelen ser especialmente efectivas/usadas en proyectos grandes y con equipos posiblemente dispersos
La arquitectura se va definiendo y mejorando a lo largo del proyecto	Se promueve que la arquitectura se defina tempranamente en el proyecto
Énfasis en los aspectos humanos: el individuo y el trabajo en equipo	Énfasis en la definición del proceso: roles, actividades y artefactos
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Se esperan cambios durante el proyecto	Se espera que no ocurran cambios de gran impacto durante el proyecto

Cuadro 4.1: Diferencia entre metodologías ágiles y tradicionales.

4.1.3 Manifiesto ágil: Principios

Los valores del apartado anterior se sustentan en 12 principios señalando así la diferencia entre un desarrollo ágil y un desarrollo clásico. Los principios son los siguientes:

- La mayor prioridad es satisfacer al cliente a partir de la entrega temprana y continua de software con valor.
- Se acepta que los requisitos cambien, en cualquier etapa del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Se entrega software funcional de forma frecuente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- Los responsables de negocio y los desarrolladores trabajan juntos durante todo el proyecto.
- Los proyectos se desarrollan en torno a un personal motivado. Hay que ofrecerles el entorno y el apoyo que necesitan, y confiar en ellos para finalizar el trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es el diálogo cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma continua.
- La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
- La simplicidad es esencial.
- Las mejores arquitecturas, requisitos y diseños surgen de equipos organizados por sí mismos.
- En intervalos regulares, el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento.

Figura 4.1: Comparación de ciclo de desarrollo ágil y tradicional.

4.2 Extreme Programming

Extreme Programming o eXtreme Programming (XP) es una metodología de desarrollo ágil que tiene como objetivo principal aumentar la productividad a la hora de desarrollar un proyecto software. Muestra una mayor importancia a los trabajos que dan un resultado directo y en los cuales se reduce la burocracia que exista en el entorno de trabajo. Hay que añadir que también añade buenas prácticas que tienen un límite mayor que la gestión de proyectos software.

XP es exitosa porque enfatiza la satisfacción del cliente. En lugar de ofrecerle todo lo que pueda desear en fecha lejana, este proceso le ofrece el software necesario a medida que lo necesita. XP les permite a sus desarrolladores responder con confianza a los requisitos cambiantes de los clientes, incluso a finales del ciclo de vida.

Investigación y estudio del entorno Tiled	
Número: 1	**Prioridad:** Media
Riesgo en desarrollo: Bajo	**Iteración:** 1
Programador responsable: Eduardo M.	**Estimación temporal:** 4 horas
Descripción: Se investiga el entorno de desarrollo de mapas basados en conjuntos de patrones o celdas.	
Validación: La tarea finaliza cuando se consiga crear un mapa adecuado para su futuro uso en el proyecto, y se hayan aprendido las distintas funcionalidades del editor.	

Figura 4.2: Ejemplo de historia de usuario.

XP prioriza el trabajo en equipo. Los gerentes, clientes y desarrolladores son socios iguales en un equipo colaborativo. También implementa un entorno simple y eficaz que permite a los equipos ser altamente productivos. El equipo se organiza a sí mismo en torno al problema para resolverlo eficientemente.

Los programadores que usan XP se comunican constantemente con sus clientes y otros programadores. Obtienen una retroalimentación al probar su software desde el primer día. Su objetivo es entregar lo antes posible el sistema a los clientes para los

cambios que estos consideren necesarios. Con estos principios los programadores extremos pueden responder valientemente a los requisitos y la tecnología cambiantes.

4.2.1 Las reglas de XP

Planificación

- Las historias de usuario tienen que ser escritas: Las historias de usuario siguen el mismo propósito que los casos de uso, pero no son lo mismo. Se usan para crear estimaciones de tiempos para la reunión de planificación de la versión. Las historias de usuario deben ser escritas por los clientes como cosas que el software debe hacer para ellos. Son similares a los casos de uso, pero no están limitados a describir una interfaz de usuario. En Fig. 4.2 se muestra un ejemplo de historia de usuario.
- La planificación de lanzamiento crea el cronograma de lanzamiento. El plan de lanzamiento se usa para crear planes de iteración para cada iteración individual. La planificación de versiones tiene un conjunto de reglas que permite que todo el personal involucrado en el proyecto tome sus propias decisiones. Las reglas definen un método para negociar un cronograma con el que todos puedan comprometerse.
- Hacer pequeñas pero frecuentes entregas. El equipo de desarrollo necesita entregar versiones iterativas del sistema a los clientes de forma frecuente. El cliente pedirá un software nuevo cada semana o dos. Al final de cada iteración, el cliente tendrá un software funcional y listo para producción.
- El proyecto se divide en iteraciones. El desarrollo iterativo agrega agilidad al proceso de desarrollo. Un ejemplo básico es dividir el cronograma de desarrollo en una docena de iteraciones, de 1 a 3 semanas de duración. Hay que intentar mantener la longitud de la iteración constante durante todo el proyecto. Esta es la clave que hace medir el progreso y que la planificación sea simple y confiable. El esquema iterativo de XP se puede observar en Fig. 4.3.
- La planificación de la iteración comienza cada iteración. Se convoca una reunión de planificación de la iteración al comienzo de cada una para producir el plan de tareas de esa iteración. Cada iteración debe durar de 1 a 3 semanas.

- Semana de 40 horas. Una propiedad importantísima dentro de la productividad es que la semana tiene 40 horas de trabajo. Si no se acaba el trabajo planificado en ese tiempo, significa que algo va mal.

Gestión

- Ofrecer al equipo un espacio de trabajo abierto y específico. La comunicación es muy importante para un equipo de XP. Es importantísimo derribar las barreras que limitan a la comunicación entre los distintos individuos que forman este equipo.
- Establecer un ritmo sostenible. Para mantener el ritmo hay que tomar un gran interés en cada iteración. Hay que obtener el software más completo, probado, integrado y listo para producción en cada iteración. El software incompleto o defectuoso representa una cantidad desconocida de esfuerzo futuro.
- Dispersar al personal. Mover a las personas para evitar la pérdida de conocimiento y la obtención de cuellos de botella. Si solo una persona en el equipo trabaja en un área determinada y esa persona tiene un problema, la productividad en el proyecto se desestabilizará bastante.
- Tener en cuenta la velocidad del proyecto. La velocidad del proyecto es una medida que mide cuanto trabajo se está haciendo en el proyecto. Para realizar esta medición, se suman las estimaciones de las historias de los usuarios que se completaron durante la iteración.

Diseño

- Simplicidad. Construir un diseño lo más simple posible para dar respuesta a las necesidades y requisitos actuales del sistema. Un diseño simple permite ejecutar todas las pruebas donde no haya lógica duplicada y que muestre las prioridades del desarrollador. También no debe tener un gran número de clases y métodos. XP permite utilizar lenguajes de modelado como Unified Modeling Language (UML). El uso de diagramas puede ayudar a mantener una dirección única en el proyecto. Los diagramas tienen que ser pequeños, por lo tanto, los elementos analizados por cada persona son limitados. Estos diagramas también deben girar en torno a los detalles importantes. La información específica está incluida

únicamente en el código, mientras que la información en alto nivel estará inmersa en los diagramas.

- Elegir una metáfora del sistema. La metáfora del sistema es en sí misma una metáfora para un diseño simple con ciertas cualidades. La cualidad más importante es poder explicar el diseño del sistema a personas nuevas sin tener que recurrir a entregarles una gran cantidad de documentación. Esta metáfora debe ser compartida entre el grupo de desarrolladores y el cliente para ayudar al diálogo. Una buena metáfora es aquella que es fácil de comprender por el cliente y contiene la información necesaria para guiar la estructura del proyecto.
- Usar tarjetas CRC para el diseño. Las Tarjetas de Clase-Responsabilidad-Colaboración sirven para diseñar el sistema como un equipo. El mayor valor de las tarjetas CRC es permitir que las personas abandonen el modo de pensamiento basado en procedimientos y aprecien más la tecnología basada en objetos. Las tarjetas CRC permiten a equipos enteros participar en el diseño. Cuanto mayor sea la participación en el diseño del sistema, mayor será la cantidad de mejores ideas.
- Refactorización siempre que sea posible. Los desarrolladores tienen a su cargo reestructurar el sistema, sin modificar su funcionalidad. De esta forma se conseguirá una eliminación del código duplicado, se reducirá el acoplamiento aumentando la cohesión y la flexibilidad. Cuando el diseño necesita ser simplificado para facilitar su mantenibilidad se realizan estas refactorizaciones.

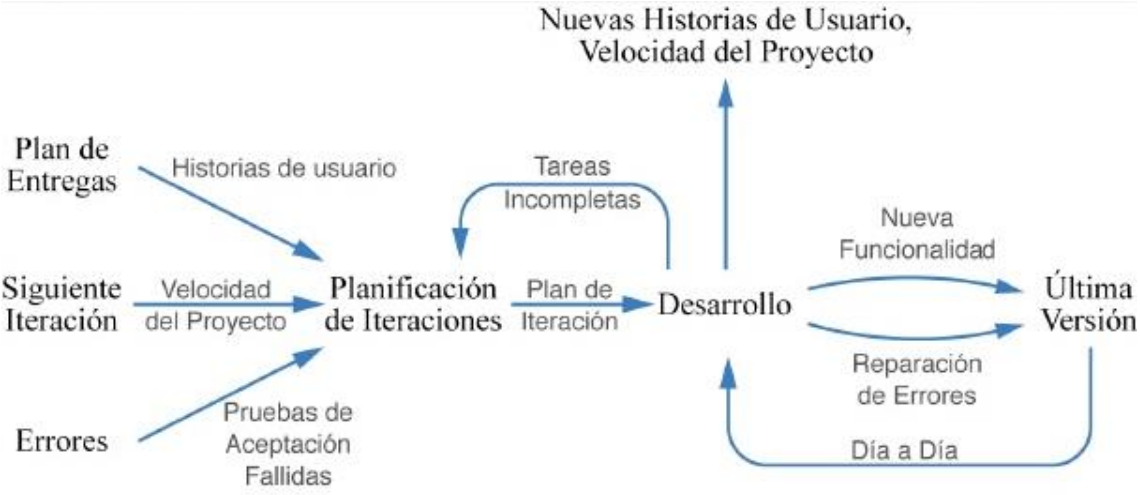

Figura 4.3: Desarrollo iterativo.

Codificación

- El cliente siempre está disponible. Esto es necesario para ayudar al equipo de desarrollo y para formar parte de este. Todas las fases de desarrollo en XP requieren comunicación con el cliente. Por lo general, se debería asignar uno o más clientes al equipo de desarrollo.
- El código se debe implementar de acuerdo con los estándares. Los estándares de código lo mantienen constante y fácil de leer y rehacer para todo el equipo.
- Implementar primero las pruebas unitarias. Cuando las pruebas unitarias se crean antes del código, resulta mucho más fácil crear el código del programa. El tiempo combinado que lleva crear una prueba unitaria y un código que apruebe esta prueba es casi lo mismo que simplemente codificar el programa directamente.
- Todo el código es programado en pares. El par de desarrolladores está formado por dos identidades. Uno es el programador del código fuente y las pruebas; mientras que el otro revisa ese código y pruebas. Por lo tanto, el segundo componente de este par será el encargado de revisar el código desde otro enfoque, incluyendo nuevas pruebas unitarias y revisando la simplicidad del diseño.
- Integración continua. Los desarrolladores deberían integrar y subir código al repositorio de código cada pocas horas. La integración continua a menudo evita esfuerzos de desarrollo divergentes, donde los desarrolladores no se comunican entre sí sobre lo que se puede reutilizar o compartir. Todos deben trabajar con la última versión de código.
- El código es una propiedad colectiva. La propiedad colectiva considera que el código pertenece al proyecto y al equipo, en lugar de a la persona que programó cada parte. Por lo tanto, cualquier participante del proyecto puede añadir nuevas funcionalidades, solucionar errores o refactorizar en cualquier parte del proyecto.

Pruebas

- El código debe tener pruebas unitarias. Una prueba unitaria es la verificación de una unidad de código determinado dentro del sistema. Las pruebas unitarias son una de las piedras angulares de XP. Las pruebas unitarias nos aseguran que un determinado módulo cumpla con un comportamiento esperado en forma aislada antes de ser integrado al sistema. En los últimos años se han creado un conjunto de frameworks llamados xUnit que permiten la automatización de las pruebas

unitarias. Además, estos frameworks permiten definir las pruebas unitarias y ejecutarlas de forma reiterada.

- El código debe pasar todas las pruebas unitarias antes de ser lanzado. En XP los programadores deben escribir pruebas unitarias para cada módulo antes de escribir el código. Después de eso, los programadores ejecutan las pruebas unitarias, las cuales tienen que tener una efectividad del 100% para que ese código pueda integrarse al sistema.
- Crear nuevas pruebas al encontrar errores. Cuando se encuentra un error, se crean nuevas pruebas para evitar que regrese. Un error en la producción requiere una prueba de aceptación para evitarlo. Crear una prueba de aceptación antes de la depuración ayuda a los clientes a definir el problema y comunicárselo a los programadores.

4.2.2 Los valores de XP

XP se basa en valores. Las reglas anteriores son una extensión o consecuencia de maximizar los presentes valores. XP es una forma de trabajar en armonía con sus valores personales y corporativos. Los valores son los siguientes:

- Simplicidad. Se desarrolla únicamente lo necesario y nada más que eso. Por lo tanto, se maximizará el valor de la inversión realizada. Se toman pequeños pasos hacia el objetivo, evitando los errores que se produzcan. Una ventaja importante es que no se invierte esfuerzo en futuros requerimientos que podrían cambiar, o simplemente no se vayan a necesitar.
- Comunicación. Todos son parte del equipo y la comunicación cara a cara es imprescindible. Todo el equipo trabaja en colaboración siempre. Esta comunicación se hace por medio de transferencia de conocimientos en reuniones de manera frecuente, entre los clientes y los desarrolladores.
- Retroalimentación. Cuando el cliente está integrado en el proyecto, se tiene la ventaja de saber su opinión en cada momento. Al llevarse a cabo breves ciclos de desarrollo, se minimiza el tener que rehacer partes que no cumplen con los requisitos aumentando así la productividad de los programadores.
- Valor. Se comunicará la verdad sobre el progreso y las estimaciones. No hay excusas para el fracaso porque en el horizonte siempre se visualiza el éxito. El equipo se adapta a los cambios cuando sucedan y el trabajo se acaba hoy y no se deja para mañana.

- Respeto. Todos los miembros de equipo se muestran y se ofrecen el mismo respeto entre ellos. Todos aportan valor, incluso si únicamente es entusiasmo. Los desarrolladores respetan la experiencia de los clientes y viceversa. Los miembros del equipo respetan su propio trabajo porque siempre se lucha por una mayor calidad en el producto y por un diseño eficiente y óptimo para la solución a través de la refactorización de código.

4.2.3 Roles en XP

En XP, el énfasis está puesto en la colaboración de todo el equipo y en la comunicación continua. A pesar de eso, se necesitan ciertos roles para que un proyecto de programación extrema funcione y las personas que asuman estos roles tengan sus responsabilidades correspondientes. Es aconsejable asignar a las personas adecuadas a cada rol en lugar de tratar de cambiar a las personas para que se ajusten a esos roles. En Fig. 4.4 se puede ver una lista de roles en XP.

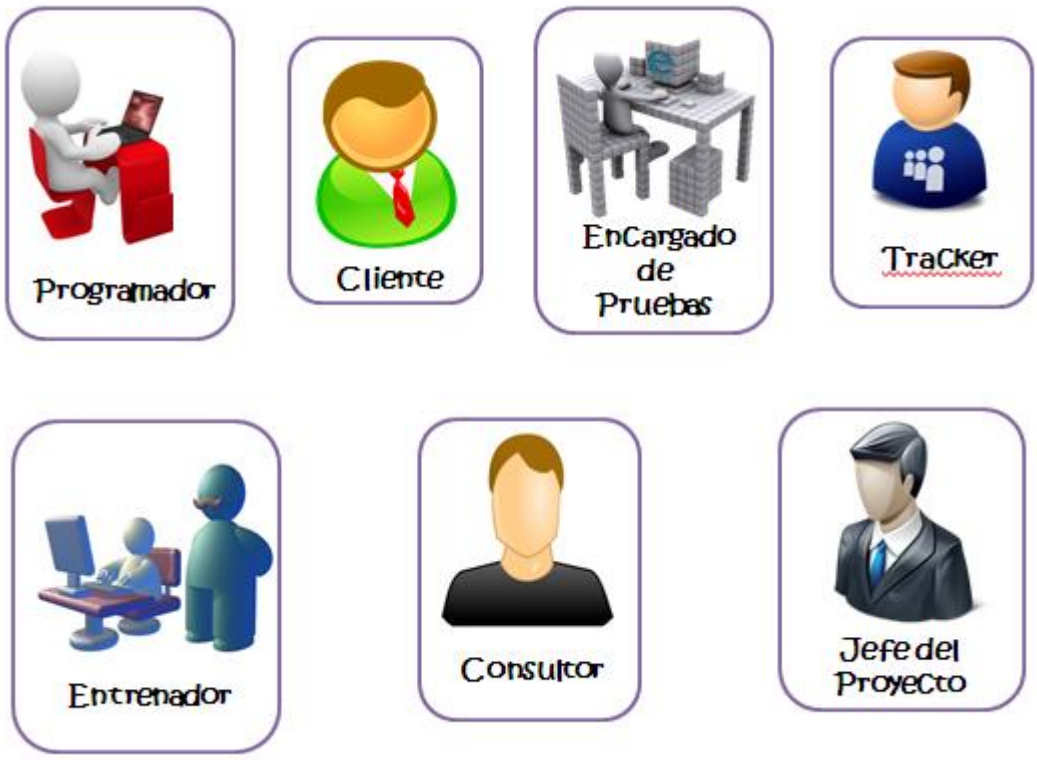

Figura 4.4: Lista de roles en XP.

Roles esenciales

- Programador. Posiblemente es el rol más importante de XP. Es el encargado de implementar el código del proyecto y las pruebas unitarias. También formula las tareas de cada historia de usuario, así como la estimación de tiempo que se requerirá en cada una.
- Cliente. En XP el rol del cliente es tan importante como el de desarrollador, ya que es el cliente el que debe saber qué programar, mientras que el desarrollador debe saber cómo programar. Es el encargado de escribir las historias de usuario y las pruebas de aceptación o funcionales. También establece las prioridades entre las historias de usuario, decidiendo así cuales se implementan en cada iteración.
- Encargado de pruebas. Es el encargado de ayudar al cliente en formular las pruebas funcionales, ejecutándolas de forma constante y difundiendo los resultados al equipo. También tiene como responsabilidad la gestión de las herramientas de soporte para pruebas.
- Tracker. Tiene como responsabilidad realizar el seguimiento del proyecto proporcionando realimentación al equipo. Es también el encargado de validar el éxito entre las estimaciones realizadas y el tiempo real gastado.
- Entrenador. Es el responsable del proyecto en general. Debería tratarse de un experto en XP, concediendo su conocimiento a los miembros del equipo en forma de guías para aplicar las normas de XP. También tiene como responsabilidad determinar la tecnología y metodologías a usar por el programador.
- Jefe de proyecto. Es el dueño tanto del equipo como de sus respectivos problemas. Debe ser experto en labores de gestión y a la tecnología asociada al proyecto. Tiene como esencial labor la coordinación del proyecto con los clientes. También es el encargado de administrar las reuniones de planes de iteración y de las agendas de compromisos.

Roles opcionales

- Consultor. El consultor es normalmente alguien exterior del equipo y debería tener como mínimo determinados conocimientos sobre algún tema necesario para el proyecto. Estas cualidades le permiten conducir al equipo del proyecto para resolver problemas determinados.
- Doomsayer. Tiene como responsabilidad transmitir al equipo los riesgos existentes al proyecto, así como la comunicación de las malas noticias. Tiene que

tener una investigación continua en cuanto a la existencia de riesgos en el proyecto, así como comunicar al equipo las posibles soluciones a estos.

4.2.4 El proceso XP

El proceso XP se basa en una serie de iteraciones. Cada iteración se considera como un ciclo de desarrollo en el que se siguen los siguientes pasos:

1. El cliente define el valor de negocio a implementar.
2. Se hace una estimación por parte del programador del esfuerzo que va a necesitar para esa implementación.
3. El cliente elige lo que quiere implementar.
4. El programador se encarga de construir ese valor de negocio.
5. Se vuelve al paso 1.

El ciclo de vida ideal de XP consiste en las siguientes fases:

Fase de exploración

En esta fase, el cliente determina de forma general las historias de usuario que son necesarias, mientras que el equipo de desarrollo empieza a entrar en contacto con las tecnologías y herramientas que serán necesarias para el desarrollo del proyecto. La fase de exploración toma de varias semanas a pocos meses, dependiendo del tamaño del proyecto y la familiaridad con estas tecnologías.

Fase de planificación

El cliente establecerá la prioridad a cada una de las historias de usuario, y a su vez, los programadores tendrán que realizar un estudio del esfuerzo necesario para cada historia de usuario. Hay un consenso entre el cliente y los programadores por la determinación del cronograma. La primera entrega no debería hacerse en más de tres meses.

Fase de iteraciones

En esta fase se realizan las diferentes iteraciones antes de la entrega del sistema. Las iteraciones no deben durar más de tres semanas. Una buena práctica es en la primera iteración establecer una estructura del sistema que sirva de referencia para el resto del

proyecto. Cuando se acabe la última iteración, el sistema debería estar listo para la fase de producción.

Fase de producción

La fase de producción consta de la realización de pruebas y validaciones antes de que el sistema sea entregado al cliente. Algunos cambios en esta fase pueden producir que se tomen decisiones para añadir nuevas características a la versión actual del sistema.

Fase de mantenimiento

Al mismo tiempo que la primera versión está en producción, el sistema tiene que estar en funcionamiento a la vez que se desarrollan nuevas iteraciones. Por lo tanto, se requiere de tareas de soporte para el cliente, produciendo así un decremento de la velocidad de desarrollo después de dejar el sistema en producción.

Fase de muerte del proyecto

Cuando el cliente no tiene más historias que incluir en el sistema, se puede decir que se ha llegado a esta fase. En esta fase el cliente puede pedir más requisitos como aquellos basados en el rendimiento y confiabilidad del sistema. Por último, se produce la documentación final del sistema y no se producirán más cambios en la arquitectura.

4.2.5 Diferencias entre XP y Scrum

Estas dos metodologías ágiles están muy alineadas, pero no son iguales, por eso se cree importante describir las diferencias entre estas. Las diferencias pueden ser bastante sutiles, pero son importantes. Estas son las diferencias más destacables:

- Los equipos de XP suelen trabajar en iteraciones que duran una o dos semanas, mientras que los equipos Scrum generalmente trabajan en iteraciones, llamadas Sprints, que duran entre dos semanas y un mes.
- Los equipos de Scrum no pueden hacer cambios en sus Sprints, es decir, cuando se completa la reunión de planificación de un Sprint, los elementos que conforman ese Sprint permanecen inmutables hasta el final del Sprint. Sin embargo, en XP los cambios en las iteraciones son mucho más concebibles.

- Los equipos de XP trabajan en un orden de prioridad estricto. Las características desarrolladas son priorizadas por el cliente y, por lo tanto, el equipo ve la necesidad de trabajar con estas características en ese orden. Por el contrario, el propietario del producto Scrum da prioridad a la acumulación de productos y el equipo es el encargado de formular la secuencia en la que desarrollarán los elementos atrasados.
- Scrum no realiza ninguna práctica de ingeniería, mientras que XP sí. Entre las prácticas de ingeniería de XP se encuentran las pruebas, el enfoque de pruebas automatizadas, la programación en pares, la refactorización, etc.

4.3 Aplicación de la metodología

Haber tomado XP como metodología de desarrollo se debe principalmente a la necesidad de una metodología para desarrollar un código complejo de implementar, así como el uso de la refactorización de código. XP también simplifica mucho los diseños, haciendo que el presente TFG sea mucho más ameno y fácil de desarrollar.

Al tratarse de una aplicación docente de verificación, XP se adapta bien a la función de este proyecto. Pero dado que este proyecto es una aplicación docente, no se han seguido a la perfección cada una de las normas de XP. En este proyecto se ha partido de una base formada a partir de los elementos de XP con los que se ha formado la metodología necesaria.

4.3.1 Asignación de los roles

Dado que el equipo de desarrollo está únicamente formado por el autor y director del presente TFG, todos los roles mostrados anteriormente se repartirán entre estos dos individuos. La asignación es la siguiente:

- El autor. Al presente redactor de estas palabras se le ha asignado el rol de programador y encargado de pruebas. Es el encargado de toda la implementación del código, así como el encargado de desarrollar todas las pruebas para verificar el software.
- El director. El director del TFG tiene asignado los roles de jefe de proyecto, entrenador y tracker. Esto es debido a que el director es experto en la tecnología asociada al proyecto, así como de la gestión de este y organizador de las reuniones

de planes de iteración. Es también el encargado de validar el éxito de las estimaciones encargadas y proporciona realimentación al equipo.

4.3.2 Iteraciones del proyecto

En este apartado se describen brevemente las diferentes iteraciones que se han llevado a cabo en este proyecto, mientras que en el siguiente capítulo se explicará detalladamente cada iteración, así como los resultados que se han obtenido. Las primeras iteraciones han durado dos semanas mientras que las últimas han durado una semana. Se ha reducido la duración al final ya que se tenía que aumentar la velocidad del proyecto para entregar el proyecto en el plazo establecido. Se realizaban las reuniones de iteración en el despacho del director del proyecto donde se especificaba qué hacer en la siguiente iteración. A continuación, se describen brevemente las distintas iteraciones:

Iteración 1

En esta iteración se realiza la primera toma de contacto con el entorno Tiled para la edición de mapas por celdas. Se estudia cómo crear un mapa para que el Framework Phaser pueda leerlo, por lo tanto el mapa creado debe ser preciso para que esta librería pueda entenderlo. Por ello, también se realiza una investigación sobre Phaser para la creación de juegos por celdas. Finalmente, cuando se han aprendido estas técnicas, se implementa un prototipo en JavaScript con un mapa que servirá como ejemplo para próximas iteraciones. En el cuadro 4.2 se pueden observar las historias de usuario de esta iteración.

Iteración 2

En la segunda iteración se implementa el algoritmo QL con el prototipo de un mundo en el lenguaje Python. Se ha tenido que investigar la creación de un mundo en Python para que se pueda visualizar el aprendizaje del agente. También se hace un estudio sobre el algoritmo QL para adaptarlo al proyecto. Una vez hecho esto se implementa el algoritmo QL en Python. Las historias de usuario de esta iteración están en el cuadro 4.3.

Iteración 3

La tercera iteración se basa en pasar la implementación de la iteración anterior a JavaScript. Por lo tanto, lo primero que hay que hacer es adaptar el mundo al algoritmo QL. Cuando se han hecho los respectivos cambios, se añade el algoritmo QL

compaginando así el Framework Phaser con el algoritmo. Las historias de usuario respectivas a esta iteración se muestran en el cuadro 4.4.

Iteración 4

En esta iteración se realiza un estudio e investigación sobre el Framework Flask para implementar el servidor REST. También se hace una investigación de la librería Bootstrap para la ayuda en la creación de las interfaces. Se crea la interfaz gráfica del inicio de sesión y del entorno de entrenamiento. Por último, se inicia la implementación del servidor en Flask, pero no se acaba hasta la siguiente iteración donde se incluye la base de datos. Se pueden observar las historias de usuario respectivas en el cuadro 4.5.

Iteración 5

Se hace una investigación sobre un Framework adecuado para integrar bases de datos a Flask. Al final, se toma la decisión de usar SQLAlchemy ya que facilita mucho el manejo de bases de datos relacionales. Se crea la interfaz del registro del usuario, así como la interfaz de carga de mapas. Para terminar el proyecto, se implementa la gestión de las bases de datos con SQLAlchemy en Flask y se termina la implementación del servidor REST con la conexión de todas las interfaces entre sí. En el cuadro 4.6 están las historias de usuario de esta última iteración.

Historias de usuario	Pronóstico
Investigación y estudio del entorno Tiled	4 horas
Investigación y aprendizaje del entorno Phaser	8 horas
Creación de mapas en Tiled apropiadamente	4 horas
Implementación del prototipo del mundo en JavaScript	12 horas

Cuadro 4.2: Historias de usuario de la iteración 1

Historias de usuario	Pronóstico
Investigación y estudio sobre el algoritmo QL	8 horas
Implementación del prototipo del mundo en Python	6 horas
Implementación del algoritmo QL con el prototipo del mundo	30 horas

Cuadro 4.3: Historias de usuario de la iteración 2

Historias de usuario	Pronóstico
Adaptar la implementación del prototipo del mundo en JavaScript	4 horas
Implementar el algoritmo QL junto con la librería Phaser	10 horas

Cuadro 4.4: Historias de usuario de la iteración 3

Historias de usuario	Pronóstico
Investigación y estudio sobre la librería Flask	6 horas
Investigación y estudio sobre Bootstrap	3 horas
Iniciar la implementación del servidor REST	15 horas
Implementar la interfaz gráfica del inicio de sesión	2 horas
Implementar la interfaz gráfica del entorno de entrenamiento	10 horas

Cuadro 4.5: Historias de usuario de la iteración 4

Historias de usuario	Pronóstico
Investigación y estudio sobre la librería SQLAlchemy en Flask	5 horas
Implementar la interfaz de registro y carga de mapas	10 horas
Implementar la base de datos con Flask-SQLAlchemy	8 horas
Acabar la implementación del servidor REST	15 horas

Cuadro 4.6: Historias de usuario de la iteración 5

Todas las historias de usuario mencionadas anteriormente se muestran detalladamente en el Anexo A del presente TFG.

Por último, en Fig. 4.5 se muestra un resumen de tareas. Se muestra el transcurso temporal en el desarrollo del proyecto, orientado por la fecha de cada reunión para la planificación de cada iteración. También se puede observar el transcurso de la consecución de las tareas.

Iteración	Sub-objetivo	Tarea	Tiempo	Fecha	Descripción
1	1	-	-	25-09-17	Reunión iteración 1
		1	4h	-	Investigación y estudio del entorno Tiled
		2	8h	-	Investigación y aprendizaje del entorno Phaser
		3	4h	-	Creación de mapas en Tiled apropiadamente
		4	12h	-	Implementación del mundo en JavaScript
2	2	-	-	9-10-17	Reunión iteración 2
		5	8h	-	Investigación y estudio sobre el algoritmo QL
		6	6h	-	Implementación del mundo en Python
		7	30h	-	Implementación del algoritmo QL en Python
3	3	-	-	23-10-17	Reunión iteración 3
		8	4h	-	Adaptar la implementación del prototipo del mundo en JavaScript
		9	10h	-	Implementar el algoritmo QL junto con la librería Phaser
4	4-5	-	-	30-10-17	Reunión iteración 4
		10	6h	-	Investigación y estudio sobre la librería Flask
		11	3h	-	Investigación y estudio sobre Bootstrap
		12	15h	-	Iniciar la implementación del servidor REST
		13	2h	-	Implementar la interfaz del inicio de sesión
		14	10h	-	Implementar la interfaz del entrenamiento
5		-	-	6-11-17	Reunión iteración 5
		15	5h	-	Investigación y estudio sobre la librería Flask-SQLAlchemy
		16	10h	-	Implementar la interfaz de registro y carga de mapas
		17	8h	-	Implementar la BBDD con SQLAlchemy
		18	15h	-	Acabar la implementación del servidor REST

Figura 4.5: Resumen de tareas.