Estudios de Informática y Multimedia

TRABAJO FIN DE CARRERA

“Metodologías de desarrollo de proyectos informáticos en entornos *web*”

2º Semestre 2010-2011

Rafael Graña Salgado

Este TFC va dedicado a las siguientes personas:

A Raquel, que siempre me apoya en las duras y en las durísimas, a la espera de que alguna vez lleguen las maduras...

A Manolo y a Feli, que siempre insistieron en que acabara unos estudios universitarios y a los que he tenido bien presentes en bibliotecas, exámenes y largas noches de prácticas.

A mis padres, que siempre han estado ahí para todo lo que he necesitado. Y todo quiere decir TODO.

A mi familia y amigos, especialmente a Luis, por no dejar de insistirme en la importancia de un título universitario. Mención especial merece también Tony, que nunca ha dejado de animarme a que entregue este proyecto.

A todos los que forman la comunidad UOC, la Universidad más importante de España en 2020 (y si no al tiempo).

Y por último a Victor, sin cuya inestimable aunque dificilmente perceptible ayuda desde la tripa de su madre jamás habría conseguido acabar esta carrera. A Isabel y Rafa IV también les debo mucho, si es que alguna vez vienen a este mundo.

¡Muchas gracias a todos!

Índice de contenido

1. Introducción

El presente TFC versa sobre las diferentes metodologías de desarrollo de proyectos informáticos.

Entendemos como **proyecto informático** un conjunto de actividades interrelacionadas, con un inicio y una finalización definidas, que utiliza unos recursos limitados (económicos, humanos y técnicos) para lograr unos objetivos determinados.

Por otro lado, entendemos la **gestión de proyectos informáticos** como un proceso de dirección y control que se concentra en la concepción, la puesta en funcionamiento, el seguimiento y la evaluación de un sistema de información particular que constituye el alcance del proyecto.

Por último, entendemos como **metodología de desarrollo de proyectos informáticos** al conjunto de técnicas y metodologías que la ingeniería del *software* pone a disposición de los responsables del proyecto para que este se implante cumpliendo en coste, plazo y calidad.

2. Objetivos

El presente TFC tiene dos objetivos principales:

- Determinar el grado de adaptación de las metodologías ágiles al desarrollo de proyectos *web* puesta en contexto con la adaptación de las metodologías clásicas a este mismo entorno.
- Realizar una clasificación de las metodologías de desarrollo de *software* según el grado de adaptación a los proyectos *web*.

Para alcanzar estos dos objetivos principales, será necesario alcanzar una serie de hitos que nos permitan aproximarnos a ellos paso a paso:

Parte 1. Estudio analítico.

En esta parte se realizará un estudio análitico que mostrará el grado de adaptación de las metodologías seleccionadas al desarrollo de proyectos en entorno *web*.

- Hito 1: Identificación de las variables que permiten medir el grado de adaptación de una determinada metodología a los proyectos *web*.
- Hito 2: Selección de cuatro metodologías ágiles y cuatro metodologías clásicas de desarrollo de software.
- Hito 3: Descripción de las metodologías seleccionadas.
- Hito 4: Clasificación de las metodologías seleccionadas según las variables obtenidas en el hito 1.

Parte 2. Caso práctico.

En esta parte 2 se desarrollará un caso práctico que permita dar respuesta desde otro punto de vista a la problemática del presente TFC.

- Hito 5: Descripción de un proyecto estándar de desarrollo *web*.
- Hito 6: División en tareas y estimación de esfuerzo atendiendo a las metodologías seleccionadas.
- Hito 7: Identificación de los hitos de seguimiento que permiten evaluar el grado de éxito de un proyecto.
- Hito 8: Planificación atendiendo a las metodologías seleccionadas.

Parte 3. Conclusiones.

3. Parte 1. Estudio Analítico.

3.1 **Hito 1.** Identificación de las variables que permiten medir el grado de adaptación de una determinada metodología a los proyectos *web*.

Son muchos los parámetros que pueden tenerse en cuenta a la hora de analizar una determinada metodología. De todos los factores posibles, para el presente TFC se tendrán en cuenta los siguientes: adaptabilidad a los cambios de requisitos, adaptabilidad a los cambios de alcance, plazo, gestión de riesgos y herramientas de gestión de proyectos. Es claro que la metodología seleccionada para afrontar un proyecto de desarrollo *web* debería ser muy tolerante a los cambios de alcance y cambios de requisitos. El plazo debe ser un factor a tener en cuenta pero no especialmente, tal y como sucede con la gestión de riesgos. Por último, dado que la complejidad tanto funcional como técnica del proyecto no resulta muy elevada, otorgamos baja importancia a las herramientas de gestión de proyectos.

En la siguiente tabla podemos observar la relevancia de los cinco parámetros preseleccionados para un proyecto *web*.

Parámetro	**Puntuación (1-5)**
Adaptabilidad a cambio de requisitos	5
Adaptabilidad a cambios de alcance	5
Plazo	3
Gestión de riesgos	3
Herramientas de gestión de proyecto	1

3.2 **Hito 2.** Selección de cuatro metodologías ágiles y cuatro metodologías clásicas de desarrollo de *software*.

Las cuatro metodologías clásicas seleccionadas serán el Ciclo de vida en cascada, el Ciclo de vida en espiral, la Metodología basada en prototipos y la Metodología de entrega por etapas.

En cuanto a las cuatro metodologías ágiles serán Programación Extrema (XP), Scrum, Crystal orange / web y Rapid Application Development. Se seleccionan las dos primeras por su gran difusión en el mundo de las TIC, Crystal orange / web por ser la metodología adaptada a *web* de una gran familia de metodologías ágiles y Rapid Application Development por ser una de las primeras metodologías ágiles en ser definidas.

3.3 **Hito 3.** Descripción de las metodologías seleccionadas.

Para comenzar se describen las metodologías clásicas seleccionadas (apartados 3.3.1 a 3.3.4) y posteriormente las cuatro metodologías ágiles (apartados 3.3.5 a 3.3.8).

3.3.1 Ciclo de vida en cascada

Esta metodología de desarrollo de proyectos fue publicada por primera vez en 1970 por Winston W. Royce. El propósito de este modelo fue introducir una metodología en el proceso de desarrollo de *software*.

En esta metodología se considera el ciclo de vida de desarrollo de *software* como un conjunto de actividades entrelazadas pero discretas, de manera que la salida de cada una de las distintas fases sirve para realimentar el modelo de cara a la siguiente fase, ya que proporciona un conjunto de requisitos para su realización.

Características generales

- El producto obtenido en cada una de las fases sirve como punto de entrada a la siguiente.
- Cada una de las fases se divide a su vez en dos. La primera de ellas consiste en la implementación de la fase propiamente dicha, y la segunda en comprobar la corrección de dicha implementación antes de generar la salida hacia la siguiente fase del modelo.
- No hay retroalimentación entre fases más allá de la expresada en el apartado anterior.
- El modelo funciona bien cuando el conjunto de requisitos se mantiene estable durante todo el proyecto.
- El modelo no funciona bien si el conjunto de requisitos varía con el tiempo.

Fases del proyecto

- Definición de requisitos.

En esta etapa se produce la comunicación entre los usuarios y el equipo de desarrollo, de manera que los primeros especifican a los segundos sus necesidades para que éstos puedan traducir esos requisitos a diseño. El entregable de esta fase es un documento con la definición formal de los requisitos.

- Diseño.

En esta fase, el equipo de desarrollo responde a los requisitos obtenidos en la fase anterior con unas especificaciones formales y funcionales que definen el funcionamiento del sistema tal y como es percibido por ellos.

En esta metodología, el diseño se concibe como un proceso iterativo, ya que la primera versión del diseño es un diseño de alto nivel del que pueden irse disgregando las diferentes piezas de *software* a desarrollar, así como los interfaces que permiten la conexión entre los distintos componentes. El diseño, así entendido, parte de un alto nivel de abstracción hacia un nivel más bajo, llegando a

generar un diseño de bajo nivel que pueda servir como entrada a la siguiente fase. De la calidad del diseño generado dependerá la calidad del producto final a desarrollar.

El entregable de esta fase será un diseño técnico detallado que permita la ejecución de la siguiente fase del ciclo de vida.

- Codificación del diseño.

En esta fase el diseño realizado en la fase anterior se convierte en algo tangible, como es el código que implementa la solución. El entregable de esta fase será dicho código correctamente escrito y documentado.

- Pruebas del diseño codificado.

Durante esta fase se comprueba que el producto funcione adecuadamente, comprobación ésta que se realiza a todos los niveles. En el nivel más bajo, los programadores deben depurar el código fuente obtenido en la fase anterior, probando cada una de las partes del sistema de manera aislada. A esto se le conoce como pruebas unitarias. Tras este proceso de prueba, el equipo de desarrollo tendrá que hacer las correcciones necesarias antes de pasar al siguiente nivel de abstracción.

En un nivel de abstracción superior se pasa a estudiar la integración entre los diferentes módulos desarrollados, probando así los interfaces que comunican las piezas aisladas que se han probado anteriormente. A esto se le conoce como pruebas integradas. Una vez más, el equipo de desarrollo tendrá que realizar las modificaciones pertinentes antes de pasar al siguiente nivel de abstracción

Una vez que las pruebas integradas se han realizado, pasaríamos a las pruebas de aceptación o pruebas de usuario. Para la realización de estas pruebas, el usuario cuenta con los requisitos originales, de manera que puede trazar el comportamiento real del sistema con el del sistema especificado. Tras estas pruebas, el usuario realizará la aceptación formal del producto.

El entregable de esta fase será un código probado y listo para la siguiente de las fases.

- Implantación.

Esta fase consiste en la puesta en producción del código resultante de la fase anterior. En el caso de que dicho código tenga impacto en sistemas que actualmente se encuentren en producción, dentro de estas fases pueden realizarse las denominadas pruebas de regresión. Estas pruebas, generalmente automatizadas, permiten probar que el funcionamiento de los sistemas afectados directa o indirectamente por el nuevo producto no se vean afectados.

El entregable de esta fase es un producto listo para ser utilizado por sus usuarios finales.

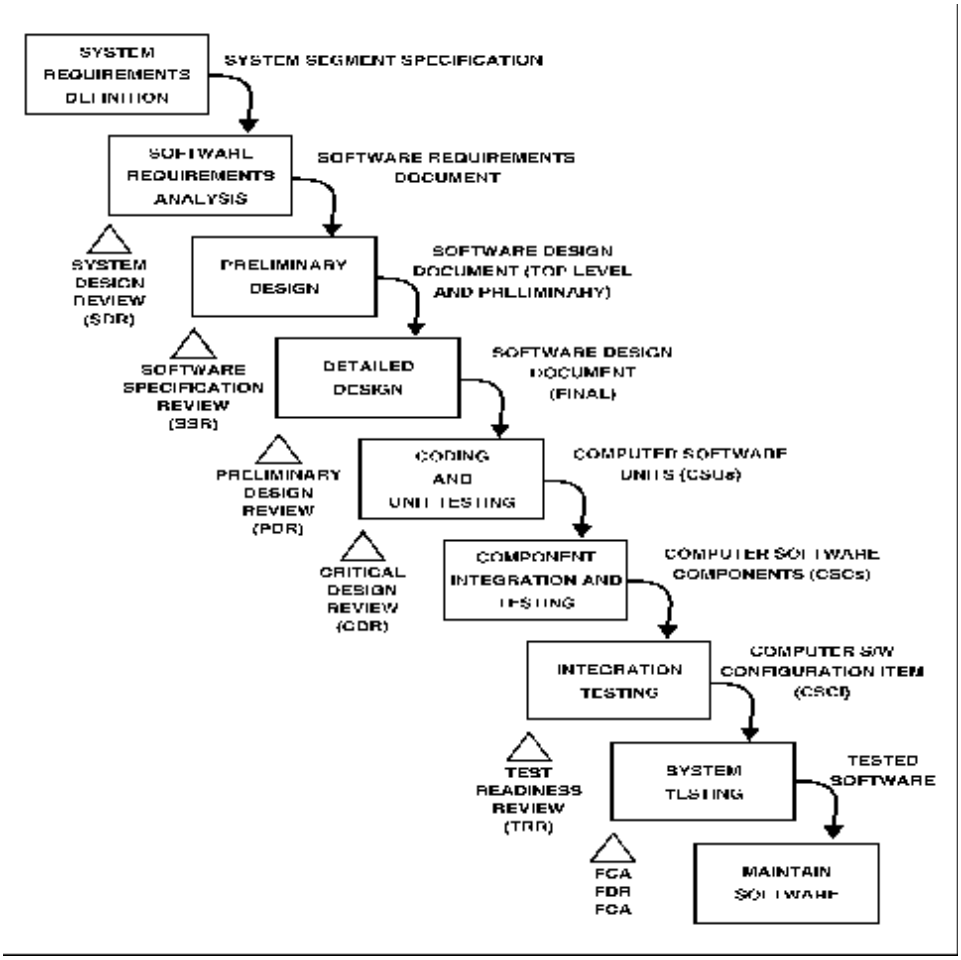
- Mantenimiento.

Una vez implantado el producto comienza la fase de mantenimiento. En esta fase se mantendrá el producto hasta que sea sustituido o eliminado.

Por otro lado, el modelo se ha visto actualizado con posterioridad a su publicación. Cabe destacar la actualización del modelo que el Departamento de Defensa de los Estados Unidos de América (DOD) ha realido en colaboración con la Administración Nacional de la Aeronaútica y el Espacio (NASA) publicada en el año 1996 y actualizada en 1999. Esta nueva versión divide el ciclo de vida en las siguientes fases:

- Conceptualización del sistema / Análisis de requerimientos del sistema.
- Análisis de requerimientos del *software*.
- Estimación paramétrica del coste del *software*.
- Diseño preliminar.
- Diseño detallado.
- Pruebas unitarias del *software* construido.
- Integración y pruebas de componentes de *software*.
- Pruebas de configuración del *software*.
- Pruebas operativas y de integración del sistema.

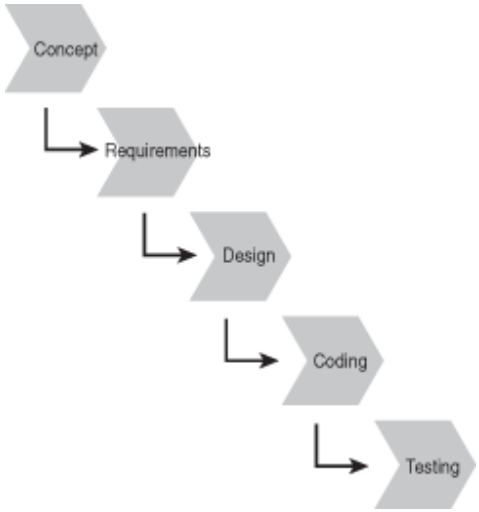
Prácticamente todas y cada una de estas fases tiene asociados unos hitos a revisar y unos entregables asociados, tal y como se muestra en el siguiente diagrama:



Así, la relación entre fases, hitos y entregables asociados es la siguiente:

Fase	Hito	Entregable
Definición de requisitos del sistema	N/A	Especificación segmentada del sistema
Análisis de requisitos del *software*	N/A	Documentación de requisitos de *software*
Diseño preliminar	Revisión del diseño del sistema	Documento de diseño del *software* (Preliminar)
Diseño detallado	Revisión de la especificación del *software*	Documento de diseño del *software* (Final)
Pruebas unitarias del *software* construido	Revisión del diseño preliminar	Unidades de *software* (CSUs)
Integración y pruebas de componentes de *software*	Revisión del diseño crítico	Componentes de *software* (CSCs)
Pruebas de integración	N/A	Elementos de configuración (CSCI)
Pruebas del sistema	Revisión de la disposición de los test	*Software* probado

La secuencia de fases de la versión clásica de este modelo puede verse en el siguiente diagrama:



Este ciclo de vida es poco tolerante a cambios en los requisitos, de modo que está diseñado para construir sistemas cuando dichos requisitos están muy claros, de manera que el alcance queda claro desde la primera fase.

Planificación del proyecto

La planificación del proyecto se basa en la división en tareas de cada una de las fases y la planificación de cada una de ellas de manera independiente. Cualquier desvío en una de las fases afecta a todas las demás, de manera que afectan al camino crítico del proyecto.

Gestión del proyecto

Esta metodología pone de manifiesto que en cada una de las fases la responsabilidad de la gestión recae en roles muy específicos.

Gestión de riesgos

- Riesgos funcionales: para mitigar los riesgos que afecten al producto final, la definición de requisitos resulta clave. Dado que la participación del usuario en el proyecto una vez que se ha cerrado la fase de definición de requisitos es nula hasta la fase de pruebas, resulta dificil mitigar el riesgo de que el producto no se corresponda con las necesidades reales del usuario final.

- Riesgos de plazo: el hecho de que la planificación del proyecto se haga en una fase tan temprana del desarrollo, hace que esta planificación pueda ser a muy largo plazo. Los riesgos en la planificación se gestionan afinando mucho en la fase de diseño los componentes a desarrollar y estimándolos de manera detallada, pero en el largo plazo esta estrategia de mitigación puede resultar insuficiente.

3.3.2 Ciclo de vida en espiral

El Ciclo de vida en espiral es un refinamiento del anterior, ya que contempla las mismas fases del desarrollo del *software*, con la salvedad de que permite varias iteraciones en dicho ciclo. Este modelo fue propuesto por primera vez por Boehm en 1988 y refinado posteriormente por Muench en 1994.

Esta metodología permite una mayor interacción con las personas que definen los requisitos, ya que en cada iteración pueden incorporarse nuevos requisitos para el refinamiento funcional de la aplicación. Esta aproximación permite una reducción en los riesgos funcionales del proyecto, como veremos más adelante.

Características generales

- Aproximación cíclica a la resolución del problema, de manera que en cada iteración aumenta el nivel de definición, y con ello se reduce el nivel de riesgo.
- Incluye hitos de control en cada una de las iteraciones para asegurar que las soluciones generadas son conformes para todos los intervinientes del proyecto.

Fases del proyecto

Con respecto a la división en fases del proyecto, en la primera versión de esta metodología eran las mismas que en el modelo en cascada. Sin embargo, en 1988, Morris propuso una nueva división en cuatro fases que presenta una visión más refinada de la metodología.

- Estudio de viabilidad.

En esta fase se decide si el proyecto va a realizarse o no. Para la toma de la decisión se incluyen en esta fase los objetivos y el alcance del proyecto, el diseño conceptual, el estudio de viabilidad de alto nivel y la formulación de la metodología. En el caso de que se decida acometer el proyecto, esta fase incluye la aprobación formal por parte de los gestores de Negocio tanto del diseño como de la metodología.

- Planificación y diseño.

En esta fase es en la que se acuerda todo lo necesario para la realización del proyecto. Esto incluye el diseño detallado, la planificación y, si procede, el coste del proyecto. Asimismo, se configura el equipo de desarrollo que afrontará el proyecto.

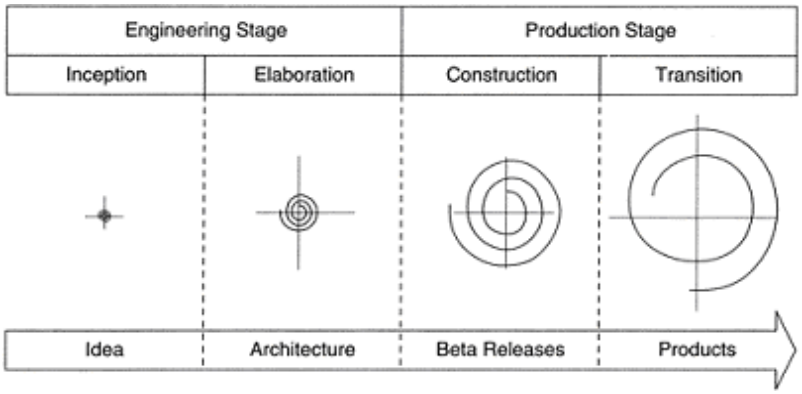
- Producción.

Esta etapa finaliza con entrega de un *software* preparado para la puesta en producción del *software* construido. Esto incluye la construcción del *software*, así como su despligue y pruebas.

- Puesta en marcha.

Al finalizar esta etapa el producto queda implantado para su utilización por los usuarios finales. La fase incluye la prueba final del sistema y la realización del plan de mantenimiento.

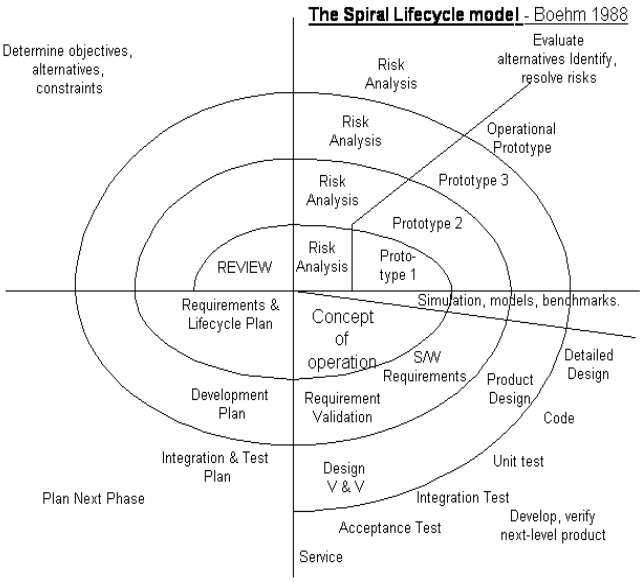
Otra versión más actual del Ciclo de vida en espiral es la ofrecida por Royce, que puede resumirse en el siguiente diagrama:



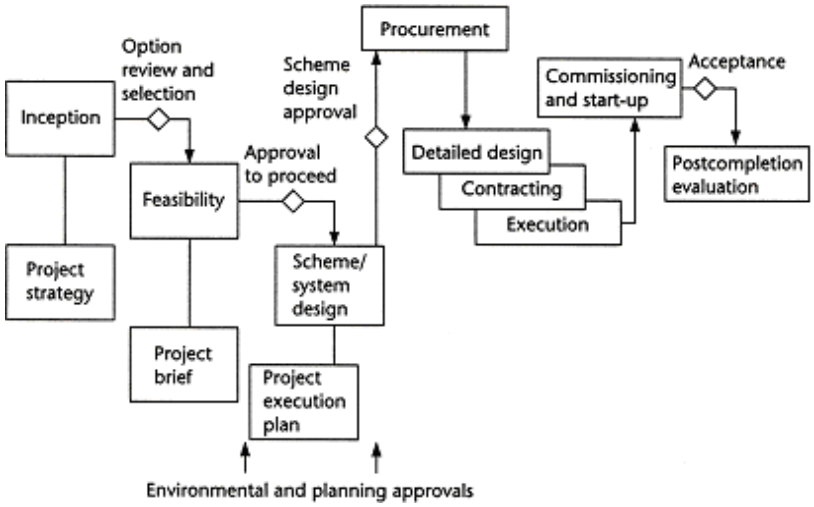
Según esta visión, el desarrollo de un proyecto pasa por dos etapas perfectamente diferenciadas, una de ellas es la de ingeniería y la otra es la de producción. Para cada una de estas etapas tenemos a su vez dos fases.

Así las cosas, en la etapa de ingeniería tenemos el origen, cuyo producto resultante es una idea. A continuación tenemos la fase de elaboración, en la que el producto resultante es una arquitectura. Por otro lado, dentro de la etapa de producción, tenemos la fase de construcción, cuyo producto resultante serán las diferentes versiones beta del producto. Por último, tenemos la fase de transición, tras la cual obtenemos productos finalizados y listos para su utilización por parte de usuarios finales.

A continuación puede verse un diagrama que resume este ciclo de vida en su vertiente más clásica:



A continuación puede verse el diagrama de flujo del modelo según la aproximación de Morris:



Planificación del proyecto

A pesar de que el modelo es iterativo, hay una tarea que se realiza una única vez durante el ciclo de vida. Esta tarea no es sino la planificación inicial del proyecto.

Gestión del proyecto

En cada una de las iteraciones que se realiza dentro del ciclo de vida, deben tenerse en cuenta las siguientes actividades de gestión, al márgen del desarrollo propiamente dicho:

- Determinar y fijar objetivos: deberán especificarse los entregables a obtener durante el desarrollo de la fase, como por ejemplo las especificaciones de alto nivel o el manual de usuario. Por otro lado, deberá tenerse en cuenta el conjunto de restricciones de la implementación.
- Análisis de riesgos.
- Planificar.

La realización de estas tres tareas de gestión en cada una de las iteraciones permite mantener el control del proyecto de una forma más óptima que en el Ciclo de vida en cascada.

Gestión de riesgos

Tal y como hemos visto en el apartado anterior, la gestión de riesgos se realiza para cada una de las distintas iteraciones del modelo. De hecho, este modelo está claramente focalizado en la localización y mitigación de los riesgos del proyecto.

3.3.3 Metodología basada en prototipos

Esta metodología se basa en la realización de un prototipo. Dicho prototipo se presenta a las personas que definen el proyecto de manera que les sirva como retroalimentación y les permita proporcionar unos requisitos suficientes para implementar el proyecto final.

Características generales

- Se trata de un modelo de desarrollo evolutivo.
- Se obtienen resultados tangibles en el corto plazo.
- El prototipo se inicia sin una versión detallada de los requisitos.
- El cliente se involucra más en el ciclo de desarrollo que en otras metodologías de desarrollo.
- El enfoque está orientado a que el prototipo sirva para generar unos requisitos completos y de calidad, en cuyo caso el prototipo se descarta. El diseño del prototipo debe ser rápido y barato, con lo cual la calidad del mismo se puede resentir.

Fases del proyecto

- Plan rápido.

Esta fase incluye la recolección de los requisitos, en la que el usuario y el equipo de desarrollo definen los objetivos generales del *software* a construir. Además, identifican aquellas funcionalidades para las que es necesario un mayor nivel de detalle. Esta identificación de funcionalidades sirve para acotar el alcance del prototipo a realizar.

Una vez que dichas funcionalidades están identificadas, hay que determinar si ese conjunto de requisitos es un buen candidato para construir un prototipo.

- Modelado, diseño rápido.

Esta fase se centra en representar aquellos aspectos del *software* que serán visibles para el usuario. Este diseño rápido será la base para la construcción del prototipo.

El diseño de un prototipo se enfoca normalmente hacia la arquitectura a nivel superior y a los aspectos de diseño de datos, en vez de hacia el diseño procedimental detallado.

- Construcción del Prototipo.

Partiendo del diseño rápido, se construye el prototipo. El objetivo del prototipo es mostrar a los usuarios cómo será el funcionamiento del sistema construido, de manera que el prototipo irá orientado fundamentalmente a las interacciones entre el usuario y el prototipo, dejando en la medida de lo posible aparcada la lógica interna.

- Desarrollo, entrega y retroalimentación.

Una vez se ha construido el prototipo teórico, en esta fase se implementa. Si el prototipo se ha diseñado con el único fin de generar nuevos requisitos de usuario, no se deberá prestar demasiada atención al desarrollo del prototipo, ya que la lógica de Negocio implementada en él se eliminará, no así la parte visual.

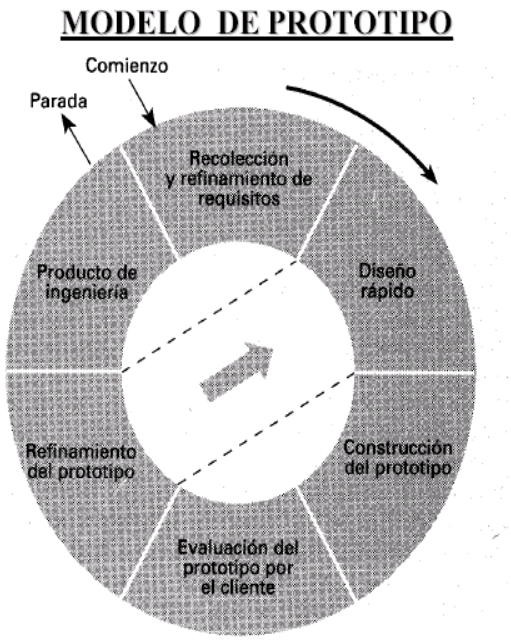
Lo más importante de esta fase es el procedimiento de retroalimentación, ya que este paso es el núcleo del método de construcción de prototipo. Es aquí donde el cliente puede examinar una representación implementada de los requerimientos del programa, sugerir modificaciones que harán al programa cumplir mejor las necesidades reales. Este conjunto de mofificaciones sugeridas realimentan el modelo de requisitos inicial para hacerlo más ajustado a las necesidades de Negocio.

- Comunicación.

Las dos fases anteriores (Construcción del prototipo y Desarrollo, entrega y retroalimentación) se iteran tantas veces sea necesario hasta que el cliente considera que el conjunto de requisitos ha quedado cerrado.

Con el prototipo finalizado y funcionando, éste se presenta al cliente. Con un producto tangible, el cliente debería ser capaz de generar nuevos requisitos que permitan una mejor adaptación del producto a las necesidades reales de la organización.

A continuación puede verse un esquema de esta metodología:



Planificación del proyecto

Por un lado, tenemos la planificación de la construcción del prototipo. El hecho de tener un alcance indeterminado a la espera de recibir nuevos requisitos tras la implementación del prototipo hace que lo único que podamos planificar inicialmente sea la realización del mismo.

Una vez el alcance del proyecto esté identificado mediante unos requisitos cerrados, tendremos que planificar el proyecto en base a alguna de las otras metodologías de desarrollo existentes. Una muy buena opción sería el Ciclo de vida en cascada, una vez que el alcance del proyecto ha quedado cerrado con la implementación del prototipo.

Gestión del proyecto

Los mecanismos de gestión del proyecto en la fase de realización del prototipo serán los siguientes:

- Reuniones de presentación de versiones del prototipo: en estas reuniones tendrá lugar la retroalimentación al equipo de desarrollo por parte de los usuarios de la aplicación. Es importante que los asistentes por parte de Negocio sean personas con potestad para definir el funcionamiento del producto final.

- Reuniones de presentación de nuevos requisitos: posteriormente a la presentación de nuevas versiones del prototipo, el usuario propondrá nuevas baterías de requisitos, que serán incluidas en el alcance de la nueva versión del prototipo o del producto final en el caso de que las distintas iteraciones del modelo se den por finalizadas.

Gestión de riesgos

- Riesgos funcionales: este modelo está orientado claramente a la mitigación de los riesgos funcionales. Con la presentación del prototipo debería producirse una alineación prácticamente perfecta entre la visión de Negocio y la visión del equipo de desarrollo. Con unos requisitos ajustados y consensuados es muy difícil de que el producto final no se ajuste a lo esperado por parte del usuario.

- Riesgos de plazo: la realización de un prototipo desechable aumenta sin duda el plazo de ejecución del proyecto, si bien es cierto que el grado de definición del mismo no sería suficiente para comenzar el proyecto, o al menos no lo era atendiendo a cualquier metodología clásica de desarrollo de proyectos informáticos.

3.3.4 Metodología de entrega por etapas

Se trata de una metodología en la que el *software* se desarrolla por etapas, generalmente desarrollando primero las funcionalidades más críticas.

Características generales

- Mayor visibilidad del proceso de desarrollo por parte de los usuarios.
- Menor periodo entre versiones del *software*.

- Alerta de los problemas más tempranamente.
- La estimación por fases reduce los errores de estimación.
- La mayor frecuencia en la entrega de producto minimiza los problemas de integración.

Fases del proyecto

- Conceptualización del *software*.

En esta fase se generan requisitos en bruto por parte de los usuarios.

- Análisis de requisitos.

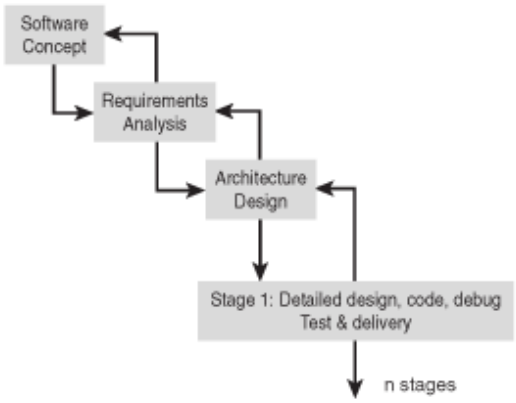
Sobre los requisitos generados por Negocio, genera un conjunto de requisitos técnicos utilizables por el equipo de desarrollo.

- Diseño de la arquitectura.

El equipo de desarrollo realizará un diseño de la arquitectura de la solución, que será común para todo el alcance del proyecto.

- Una iteración de cada una de las siguientes fases para cada entrega.
    - Diseño detallado
    - Codificación
    - Pruebas unitarias
    - Pruebas integrales
    - Entrega

El siguiente diagrama muestra de manera esquemática las diferentes fases que implementa esta metodología:



Planificación del proyecto

Antes del comienzo del proyecto se realiza una planificación inicial del mismo. Posteriormente, para cada una de las entregas programadas se realiza una planificación *ad hoc*.

Gestión del proyecto

El equipo de desarrollo trabaja sin relación con el usuario hasta la entrega de la fase. En ese punto el usuario valida que la entrega se corresponde con las funcionalidades solicitadas a través de los requisitos.

Gestión de riesgos

- Riesgos funcionales: para la mitigación de los riesgos funcionales en esta metodología se realizan también entregas planificadas de la parte funcional, si bien éstas no tienen por qué coincidir con las entregas de desarrollo planificadas, ya que no todas tienen que tener visibilidad para el usuario final.
  Aún con esta estrategia de mitigación, este es el mayor riesgo de la presente metodología. Una vez que la primera entrega está desarrollada y se presenta al usuario, no es extraño que éste genere nuevos requisitos a desarrollar. Esto hace desaconsejable la utilización de esta metodología cuando el conjunto de requisitos no esté lo suficientemente acotado.

- Riesgos de plazo: esta metodología está claramente orientada a la reducción de los riesgos de plazo. En lugar de realizar una única planificación global, se planifican diferentes entregas a lo largo de la vida del proyecto, de manera que las desviaciones son menos probables y más fácilmente gestionables.

3.3.5 Programación extrema (XP)

Se trata de una metodología de desarrollo de proyectos desarrollada en 1999 por Ken Beck. Dicha metodología se engloba dentro de las denominadas metodología ágiles, y supone un cambio en el foco de atención con respecto a las dos metodologías vistas anteriormente.

Concretamente, la metodología de Programación extrema pone el énfasis en la adaptabilidad en lugar de en la previsibilidad. Por ello, esta metodología contempla los cambios de requisitos sobre la marcha, pues considera que es algo natural dentro de los procesos de desarrollo de *software*.

Características generales

- Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- Los roles dentro del equipo de trabajo desaparecen. La forma de trabajar es situar a dos personas en una pantalla al mismo tiempo realizando todo el abanico de tareas de desarrollo.
- Programación en parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que la mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata.
- Frecuente integración del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- Corrección de todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.

- Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
- Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La metodología de Programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

Prácticas específicas de la metodología

- El juego de la planificación: se trata de una manera de planificar que permite definir de manera sencilla el alcance de cada una de las entregas.
- Entregas pequeñas: las entregas a realizar serán lo más pequeñas posibles siendo lo suficientemente completas. El objetivo es poner una versión del sistema en producción lo más rápidamente posible.
- Metáfora: la metáfora ofrece al equipo de desarrollo una visión de alto nivel del funcionamiento del sistema final, a modo de guía.
- Diseño simple: el sistema debe diseñarse de la manera más sencilla posible.
- Test: primeramente, los desarrolladores realizan pruebas unitarias que les permitan validar las funcionalidades desarrolladas. Posteriormente, los usuarios realizan pruebas para cerrar las funcionalidades desarrolladas.
- Refactorización: las modificaciones en el código que no afecten al comportamiento del mismo pueden realizarse en cualquier momento del desarrollo.
- Programación en parejas: el código se desarrolla por parejas de programadores trabajando juntos en la misma máquina.
- Propiedad colectiva: todo el equipo de desarrollo está autorizado a modificar cualquier elemento de código en cualquier momento.
- Integración contínua: la integración puede realizarse varias veces al día. En general, se realizará cada vez que una tarea esté finalizada.
- 40 horas semanales: marcar como regla no programar más de 40 horas semanales. Si extraordinariamente se realizan horas extra, nunca en semanas consecutivas.
- Cliente presencial: al menos un representante de Negocio debe estar trabajando con el equipo de trabajo a jornada completa, con el fín de poder resolver dudas de manera inmediata.
- Estándares de codificación: deben definirse unos estándares de codificación, en cuyo mantenimiento colaborarán todos los miembros del equipo.

Fases del proyecto

El proyecto se divide en diferentes entregas. Cada una de ellas contempla un conjunto de fiuncionalidades completo y con ello un flujo de desarrollo también completo.

En el siguiente esquema puede verse el flujo de trabajo de esta metodología:

Planificación del proyecto

En Programación extrema la responsabilidad de la planificación recae fundamentalmente en los recursos de Negocio involucrados en el proyecto, siendo responsables de las siguientes tareas de definición:

- Alcance del proyecto: las personas de Negocio son las encargadas de valorar hasta qué punto un sistema o funcionalidad resulta útil y como debe ser su comportamiento.
- Prioridad: las personas deberán ser capaces de determinar qué componentes del sistema serán los más valiosos para la organización, de manera que sean implementados antes por el equipo de desarrollo.
- Composición de las entregas: determinarán qué funcionalidades deben agruparse para su despliegue en producción.
- Fecha de las entregas: las personas de Negocio serán las encargadas de definir en qué fechas deben estar listas las funcionalidades en el entorno correspondiente.

Por otro lado, los técnicos serán responsables de las siguientes tareas de definición:

- Estimaciones: serán los encargados de determinar el tiempo que llevará implementar una determinada funcionalidad.
- Informar sobre las consecuencias de las decisiones de Negocio: entre sus funciones se encuentra la de asesorar al equipo de Negocio de las implicaciones que sus decisiones tendrán en los sistemas y/o en los procesos.
- Proceso: se encargarán de constituir el equipo de trabajo y orientarles hacia la consecución del objetivo común.
- Planificación detallada: determinarán, para cada una de las entregas, el orden de realización de las tareas que conlleve.

Una vez distribuidas las responsabilidades entre los diferentes roles, veamos qué características debe cumplir la planificación en Programación extrema:

- Horizonte de planificación: será cercano, situándose siempre en el final de la próxima entrega.
- Planificación horizontal: la planificación no será jerárquica, sino que todos los miembros del equipo deberán tomar su propia responsabilidad en lo que a planificación se refiere.
- Estimación de las tareas: será realizada por aquellas personas encargadas de implementarla.
- Ignorar las dependencias entre partes: la planificación debe realizarse como si las distintas piezas de *software* fueran totalmente independientes.
- Planificación por prioridades: las operativas más valiosas serán programadas para las primeras fases del desarrollo.
- Planificación de la primera entrega: idealmente, la primera entrega debería situarse entre dos y seis meses después de la fecha de inicio del proyecto. En general, deberá ser lo suficientemente larga como para que el despliegue en producción de la funcionalidad aporte valor al Negocio y lo suficientemente corta como para que no se dilate demasiado en el tiempo.

En esta metodología, la planificación de cada una de las entregas se realiza en tres fases:

1. Exploración.

En esta fase los distintos participantes en el proyecto deben definir lo que el sistema debería hacer. A su vez se divide en tres movimientos:

- Describir una tarea: Negocio se encarga de definir el comportamiento de una determinada pieza dentro del sistema.
- Estimación de la tarea: el equipo de desarrollo se encarga de estimar el tamaño de dicha pieza del sistema.
- Dividir la tarea: en el caso de que cierta parte de la pieza aporte más valor que el resto, la tarea se puede dividir.

2. Acuerdo.

El objetivo de esta fase es decidir el alcance y la fecha de la siguiente entrega. Negocio hará la propuesta y el equipo de desarrollo se encargará de comprobar si puede llegar a la fecha requerida con el alcance propuesto. A su vez está dividida en las siguientes subfases:

- Ordenación por valor: las piezas se ordenan en tres grupos. En el primero de ellos se incorporan aquellas funcionalidades sin las cuales el sistema ni funciona. En el segundo grupo de funcionalidad se sitúan las que aportan un valor considerable a la organización. En el último de los grupos se sitúan las funcionalidades que no resultan clave, sino deseables.
- Ordenación por riesgo: las piezas se ordenan, igual que en el caso anterior, en tres grupos. En el primero de ellos se sitúan las funcionalidades que se pueden estimar de manera precisa. En el segundo aquellas funcionalidades que pueden estimarse razonablemente bien. En el último grupo se sitúan las funcionalidades que no pueden estimarse en absoluto.

- Determinación de la velocidad: el equipo de desarrollo determinará el esfuerzo que puede realizar por mes de trabajo.
- Determinación del alcance: esta fase puede realizarse de dos maneras, bien ajustar el alcance en base a la fecha de entrega requerida o bien calcular dicha fecha en base al alcance deseado.

3. Seguimiento.

El propósito de esta fase es actualizar el plan en base a los resultados de la fase anterior. Esta fase se divide a su vez en cuatro subfases.

- Iteración: tiene una duración de una a tres semanas. En cada iteración Negocio localiza las piezas más valiosas para ser implementadas. Debe tenerse en cuenta que en el caso de la primera iteración deben elegirse piezas suficientes para completar al menos una funcionalidad de principio a fin.
- Recuperación: en el caso de que se haya sobreestimado la capacidad de esfuerzo del equipo de desarrollo y las fechas no se estén cumpliendo, en esta subfase es posible reconsiderar con Negocio un nuevo alcance para la iteración.
- Nueva pieza: en el caso de que Negocio considere oportuno incluir una nueva pieza en la iteración, ésta se estimará y en caso de que se considere oportuno se incluirá en la iteración, teniendo en cuenta el cambio de alcance.
- Re-estimación: en el caso de que el equipo de desarrollo no esté conforme con el plan de desarrollo, éste se podrá modificar por parte del equipo de desarrollo para ajustarlo a la realidad.

Gestión del proyecto

Como hemos podido ver en la introducción a Programación extrema, el proyecto se divide en una serie de mejoras sucesivas a las que denominaremos entregas. Cada una de las entregas llevará implícita una nueva planificación, en definitiva un nuevo ciclo de desarrollo completo.

La gestión de un proyecto que se afronte desde Programación extrema no recae ni en un único gestor de proyecto ni tampoco se disgrega entre todo el equipo del proyecto. Así las cosas, el gestor del proyecto será la persona que se encargue de resaltar las tareas que deben ser realizadas, no de repartirlas entre los miembros del equipo.

Otro de los principios de la gestión de proyectos es que la relación entre el gestor y el equipo ha de estar basada en la sinceridad y el compromiso, de manera que el gestor sea el primer interesado en que el equipo consiga generar un trabajo de calidad orientado a la consecución del objetivo común. Para ello, tendrá que ser honesto a la hora de realizar estimaciones de esfuerzo así como con el cliente. En lo relativo al cliente, el gestor será el responsable de asegurar que el entorno se adapte a la metodología de Programación extrema, ya que en ocasiones el modelo choca con la forma de trabajar de la empresa a la que se le presta el servicio.

En lo referente a los mecanismos de gestión del proyecto, el gestor lo hará de la manera más ágil posible, evitando largas reuniones o la elaboración de más documentación de la estrictamente necesaria para el seguimiento del proyecto.

Gestión de riesgos

- Riesgos funcionales: el mecanismo de mitigación de riesgos funcionales de esta metodología es doble. Por un lado, la inclusión del usuario en todas las fases del desarrollo resulta clave para asegurar que éste se ajusta a los requerimientos de Negocio. Por otro lado, la realización de diversas entregas completas en funcionalidad hace que en cada ciclo no sólo tengamos retroalimentación por parte de las personas que han realizado los requisitos de la aplicación, sino por parte de los usuarios finales.

- Riesgos de plazo: en esta metodología de desarrollo las planificaciones se realizan a corto plazo. Este hecho, unido a la inclusión del usuario en todas las fases del desarrollo, hace que los cambios en la planificación puedan ser tenidos en cuenta en cuanto se producen. Obviamente, el riesgo de este método de trabajo es que puede ser que el producto nunca resulte del todo satisfactorio para Negocio y los proyectos se eternicen en el tiempo. La flexibilidad del modelo puede verse de esta manera como un claro riesgo que afecta al plazo.

3.3.6 Scrum

Scrum es una metodología de desarrollo de proyectos de las denominadas ágiles. Fue formulada por primera vez por Takeuchi y Nonaka en 1986, pero hasta 1991 no adquirió su denominación actual. Su denominación tiene origen en la voz inglesa *scrum*, que significa melé. Dicha melé hace referencia a la jugada de rugby en la que todo el equipo unido avanza junto pasándose la pelota de unos a otros.

El foco de Scrum es la productividad y la velocidad del desarrollo, asumiendo que durante un proyecto los requisitos de Negocio pueden cambiar y que las planificaciones a más de 30 días vista no son posibles.

Características generales

- Es un modelo iterativo e incremental.
- Define una serie de roles relacionados con el proyecto. Estos roles son el *ScrumMaster*, que sería un rol análogo al jefe de proyecto clásico, el *ProductOwner*, que representa a los clientes del proyecto (ya sean internos o externos) y el *Team*, que representa al equipo de desarrollo.
- Dichos roles se dividen según su nivel de relación con el proyecto, dependiendo de si están comprometidos o simplemente involucrados en el proyecto.
- El proyecto se divide en *Sprints*, que son periodos de tiempo entre 15 y 30 días a cuya finalización se dispone de un conjunto de funcionalidades desarrolladas entregables.
- Los requisitos de cada *Sprint* se obtienen del *Product Backlog*, que no es sino el conjunto de requisitos de alto nivel de la aplicación ordenados por prioridad.
- El conjunto de requisitos (obtenidos del *Product Backlog*) que se aborda en cada *Sprint* se define en una reunión a tal efecto, conocida como *Sprint Planning*.

- Durante la ejecución de un *Sprint* los requisitos quedan congelados.

Fases del proyecto

Esta metodología contempla la división de cada uno de sus *Sprint* en tres fases:

- Prepartido (pregame).

Esta fase incluye dos subfases: planificación y arquitectura. La planificación de la nueva iteración está basada en el *backlog* pendiente, así como su estimación de plazo y coste. Esta planificación dependerá de si el *Sprint* pretende añadir nuevas funcionalidades a un sistema ya existente o crear un nuevo sistema o subsistema, ya que en este último caso habrá que realizar la construcción conceptual del sistema. En ambos casos será necesario realizar el análisis.

Por otro lado, la subfase de aquitectura incluye tanto el diseño de las modificaciones en la arquitectura como el diseño de alto nivel.

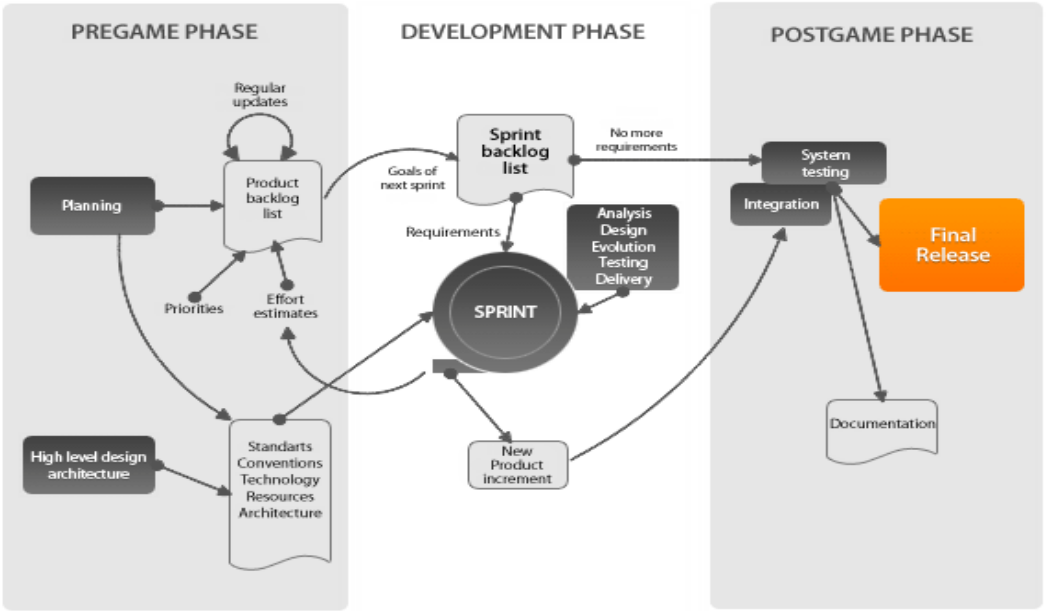
- Partido (game).

Esta fase contiene todo lo relativo al desarrollo del *Sprint* propiamente dicho. Recibe de la fase anterior el *backlog* del *Sprint*, que será un subconjunto del *backlog* del producto. La fase del partido incluye análisis, diseño, evaluación, pruebas y la entrega correspondiente.

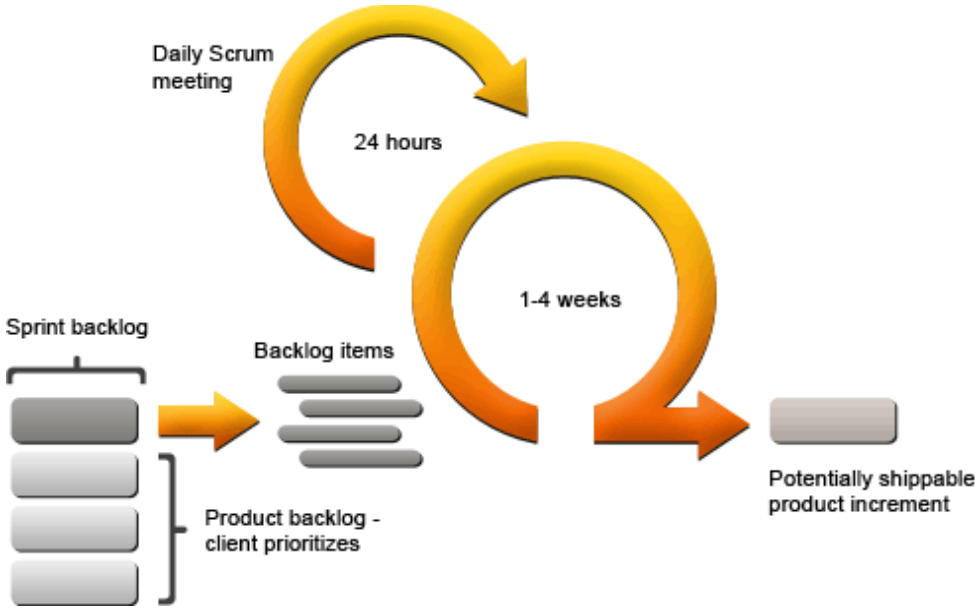
- Pospartido (postgame).

Es la fase de cierre del *Sprint*. En ella se incluyen las pruebas de aceptación del *Sprint* actual, la entrega de la documentación correspondiente, la reunión de Revisión del *Sprint* y la preparación del siguiente *Sprint* en caso de que lo haya.

A continuación puede verse la división en tres fases del ciclo de desarrollo:



El siguiente esquema muestra el ciclo de vida de esta metodología:



Planificación del proyecto

Cada uno de los *Sprint* que configuran el proyecto tiene su propia reunión de planificación denominada *Sprint Planning Meeting*. En ellas se realizan las siguientes acciones:

- Identificar el alcance de la fase, es decir, el trabajo que será acometido.
- Preparar el *Sprint Backlog* (conjunto de requisitos necesarios para poder afrontar el desarrollo) antes del inicio de la fase.
- Poner en común el volumen de trabajo para acometer el alcance definido.

Acorde a la metodología, esta reunión no deberá durar en ningún caso más de ocho horas.

El modelo de planificación por *Sprint* hace que las planificaciones sean a corto plazo, lo que evita grandes desvíos en la fecha de entrega.

Gestión del proyecto

Esta metodología incluye varios mecanismos de gestión del proyecto. Por un lado, tenemos los denominados Daily Scrum, que consisten en una reunión diaria en la que se siguen las siguientes reglas:

- La reunión debe comenzar puntualmente. Se podrá llegar a pactar castigos para aquellas personas que no acudan a la convocatoria a la hora requerida.
- Aunque todos los intervinientes en el proyecto pueden acudir, sólo los roles “cerdo” tienen derecho a la palabra.
- La reunión tendrá una duración fija de quince minutos.
- Todos los asistentes a la reunión se mantendrán en pie durante la celebración de la misma, para fomentar que la reunión no se extienda más allá de los quince minutos establecidos.

- La reunión se celebrará en el mismo lugar y a la misma hora durante toda la duración del *Sprint*.

El objetivo de esta reunión será la respuesta a tres preguntas fundamentales formuladas a todos los miembros del equipo con rol “cerdo”:

- ¿Qué has hecho desde el último Daily Scrum?
- ¿Qué es lo que has planteado hacer hoy?
- ¿Qué impedimentos has tenido para completar tu tareas?

Además de este mecanismo de gestión existe una segunda reunión diaria denominada Scrum del Scrum. Esta es una reunión de trabajo en la que se coordina el trabajo de los diferentes equipos, fundamentalmente para tratar asuntos que conciernen a todos los grupos de desarrollo, como la integración entre las diferentes piezas del proyecto. Por cada uno de los grupos de trabajo acude un único representante.

La agenda de la reunión incluirá el contenido del Daily Scrum, además de las siguientes preguntas:

- ¿Qué ha hecho tu equipo desde la última reunión?
- ¿Qué hará tu equipo hasta la próxima reunión?
- ¿Qué impedimentos ha tenido tu equipo para completar sus tareas?
- ¿Hay algún trabajo planificado en tu equipo que pueda impactar en el trabajo del resto de equipos?

Además de estas dos reuniones diarias, existen otras dos reuniones muy importantes para la gestión del proyecto. Ambas tienen lugar de manera única por *Sprint*. La primera de ellas es la Reunión de Revisión del *Sprint*, que tiene el siguiente contenido:

- Revisar el trabajo completado y no completado.
- Presentar el trabajo completado (a esto se le denomina demo).
- La reunión tendrá una duración máxima de cuatro horas.

La segunda reunión es la denominada Retrospectiva del *Sprint*. Esta reunión tiene una duración fija de cuatro horas y en ella todas las personas que han participado en el *Sprint* tienen ocasión de dar sus impresiones sobre lo acontecido en el mismo. Es la reunión de cierre del *Sprint*, y su objetivo es generar un catálogo de lecciones aprendidas a tener en cuenta en sucesivos *Sprint*.

Gestión de riesgos

- Riesgos funcionales: como hemos visto en las características de esta metodología, los requisitos quedan congelados mientras se está desarrollando un *Sprint*, cuyo plazo de ejecución no superará en ningún caso los treinta días. Así las cosas, cuando ha finalizado la ejecución del *Sprint*, tiene lugar la reunión de revisión. En ese punto es cuando el usuario puede validar formalmente si el producto entregado se corresponde a lo esperado o no, aunque haya podido asistir a las reuniones diarias para comprobar el estado del desarrollo.

De manera que, en esta metodología, no existen cambios de alcance dentro del *Sprint* y en caso de que se necesiten modificaciones funcionales éstas se realizarían en *Sprints* posteriores.

- Riesgos de plazo: en esta metodología de desarrollo las planificaciones se realizan a corto plazo, no más de treinta días. Además, se realiza un seguimiento diario de las tareas. Estas son las dos metodologías de mitigación de los riesgos de plazo.

3.3.7 Crystal orange / web

En realidad se trata de una familia de metodologías de desarrollo de *software*. Dicha familia se segmenta en varias metodologías, de manera que queda dividida en las diferentes bandas que un cristal puede emitir (blanco cristalino - crystal clear -, amarillo, naranja, marrón, azul y violeta).

Toda esta familia de metodologías posee una serie de denominadores comunes: enfocadas claramente a la comunicación, centradas en las personas, ligeras en la producción de trabajo y auto-adaptativas. En apartados siguientes se verán en detalle estas características:

- Se trata de una metodología ágil.
- Colaborativa: Para utilizar Crystal debe asumirse que el desarrollo de *software* es un juego de equipo.
- Enfocada en la comunicación: En esta metodología la comunicación cobra un papel mayúsculo, dado que considera que el enriquecimiento de la comunicación entre los miembros del equipo acelera el desarrollo del producto. Por ello el equipo de trabajo deberá estar junto en una misma ubicación física.
- Centrada en las personas: esta metodología sume que el motivo de que exista el *software* es para que sea utilizado por personas.
- Ligera en la producción del trabajo: utilizando esta metodología el primer objetivo es construir *software* útil y que funcione, pero el segundo objetivo será prepararse para posteriores implementaciones. Para conseguir que el código funcione y reducir el tiempo destinado a la corrección de errores, Crystal incorpora como buena práctica la compilación y ejecución frecuente del producto en construcción.
- Auto-adaptativas: la metodología se adapta a los diferentes cambios ambientales que pudieran producirse, tales como el tamaño del equipo. Las asunciones que son buenas para equipos de cuatro personas no tienen porque serlo si el equipo aumenta hasta los veinte miembros.

Por ello, la metodología en si misma se considera una metodología base para la realización del proyecto, pero debe ir evolucionando con el mismo.

- Alta tolerancia: esta metodología asume que el bagaje de las personas que forman el equipo de trabajo es diferente, de manera que es tolerante a la utilización de préstamos de otras metodología ágiles, tales como PSP o XP.

Además de este conjunto de características generales para toda la familia de metodologías, a continuación se incluyen las características de la metodología Crystal recomendada para el desarrollo *web:* la metodología Crystal orange / web.

- Entrega constante: dado que una de las características principales del desarrollo *web* es el cambio continuo, esta metodología considera el proyecto de desarrollo como una sucesión de entregas sin solución de continuidad, de manera que a cada una de las entregas le sigue otra hasta que la aplicación *web* se extinga.
- Ciclos de desarrollo de dos semanas: en general el ciclo de desarrollo será de dos semanas, aunque este periodo podrá incrementarse hasta las cuatro semanas. Esto supone que cada cuatro semanas debe generarse un entregable estable de producto.
- Reuniones de trabajo al cierre de cada ciclo: a la finalización de cada uno de los ciclos el equipo se reune para actualizar los catálogos de problemas y buenas prácticas. Como resultado de la reunión se genera una lista de aspectos a tener en cuenta.
- Priorización de funcionalidades: el equpo debe ponerse de acuerdo para realizar en primer lugar aquellas funcionalidades más valiosas para el producto final.
- Asignación individualizada de funcionalidades: una vez que las principales funcionalidades están identificadas, éstas se reparten entre los diferentes miembros del equipo, de manera que estos tengan una priorización clara a corto plazo de las tareas a realizar.
- Seguimiento diario: cada desarrollador será responsable de escribir diariamente en una pizarra o similar el estado de sus tareas, de manera que la persona responsable de cada uno de los diferentes desarrollos pueda saber el grado de avance de cada una de las tareas repartidas entre los miembros del equipo. Una vez que el seguimiento diario se haya producido, los responsables no tienen permitido realizar ningún tipo de seguimiento hasta el día siguiente.
- Tiempo de enfoque: existe una franja horaria de unas dos horas en la que no pueden tener lugar reuniones y en la que los teléfonos deberán estar en modo silencio. Dicho periodo de tiempo será el inmediatamente posterior a las reuniones de seguimiento diario.
- Minimización de los errores: cuando una pieza de *software* se haya desarrollado, primeramente será probado por otro desarrollador para corroborar que cumple con los estándares de calidad. Una vez que el otro desarrollador da el visto bueno, se realizarán las pruebas unitarias de *software* asociadas a la pieza desarrollada. Para que una pieza de desarrollo se integre con el resto, además de lo expresado anteriormente, deberán incluirse los casos de prueba de integración y una nota de algún otro desarrollador del equipo autorizando al despliegue del desarrollo en el entorno de integración.

Fases del proyecto

En este apartado nos referiremos a la división en fases de un proyecto desarrollado con la metodología Crystal orange / web:

- Un responsable de negocio escribe el caso de uso de negocio y el caso de uso del sistema en un mismo documento. El caso de uso de negocio describe las funcionalidades del nuevo sistema propuesto. Para ello realizará especial hincapié en los procesos que requieran interacción humana.
- El documento obtenido en la fase anterior es utilizado por el grupo de desarrollo para estimar el trabajo necesario para construir las funcionalidades descritas.

- Los ejecutivos de negocio revisan tanto los casos de uso como la estimación obtenida por el equipo de desarrollo, así cómo el valor esperado del producto final. Una vez este conjunto está revisado y aprobado se continúa con el desarrollo del proyecto.
- A continuación los analistas de negocio definen el flujo básico de la aplicación, tras lo cual las pantallas son diseñadas por el departamento encargado de la maquetación (P.E. Marketing).
- El analista de negocio detalla los casos de uso, así como los flujos de datos. Todo esto se envía al equipo de desarrollo.
- Una vez que el código está desarrollado y ha pasado la validación descrita en apartados anteriores, se entrega al equipo de pruebas de integración, que se encarga de reportar cualquier problema al equipo de desarrollo.
- Una vez que el desarrollo está puesto en producción, se constituye un equipo denominado SWAT que se encarga de solucionar cualquier incidencia que pueda surgir con la aplicación. Este equipo de solución rápida de problemas va rotando cada dos ciclos.

A continuación puede verse las distintas actividades a realizar en un proyecto realizado siguiendo esta metodología:

Planificación del proyecto

De cara a la planificación del proyecto, la metodología Crystal orange / web da una serie de puntos básicos a considerar:

- Definición del proyecto: debe establecerse la necesidad del proyecto así como sus requisitos asociados. Además, se hace necesario el establecimiento del alcance del proyecto y conseguir el compromiso de todos los implicados antes del comienzo de la ejecución del mismo.
- Objetivos claros y alcanzables: se definen las actividades y su secuenciación, así cómo los recursos necesarios para completar el proyecto con éxito. Esto incluye la planificación y los hitos necesarios para asegurar que el proyecto se completa a tiempo y en presupuesto.
- Coordinación de recursos: se identificará y coordinará a los intervinientes en el proyecto para alcanzar la calidad, alcance, tiempo y coste objetivo del proyecto.
- Comunicación, Control y Monitorización: regularmente se medirá y monitorizará el avance del proyecto para identificar desviaciones a tiempo de poder tomar acciones correctivas antes de que el proyecto se vea impactado. Por otro lado, se comunicará el estado del proyecto para mantener informados a los intervinientes de los avances en el mismo.

Gestión del proyecto

La gestión del proyecto se realiza a través de la actualización diaria por parte del equipo de trabajo del estado de sus tareas. A continuación, y una sola vez por día, el equipo reporta a los responsables el estado de sus tareas. En el momento que finalizan las diferentes reuniones de seguimiento, el equipo dispone de una franja de dos horas en la que no será interrumpido y que sirve para enfocar adecuadamente el trabajo planificado para el día.

Gestión de riesgos

- Riesgos funcionales: en esta metodología, los roles de analista funcional y analista técnico están fusionados en lo que se conoce como analista de negocio. Esta persona define el funcionamiento de la aplicación llegando hasta los flujos de datos.

- Riesgos de plazo: como hemos visto en apartados anteriores, esta metodología incluye un mecanismo de comunicación, control y monitorización que permite hacer un seguimiento del grado de avance del proyecto y poder realizar acciones correctivas o preventivas en el caso de que se produzca alguna desviación.

3.3.8 Rapid Application Development (RAD)

Se trata de una metodología diseñada inicialmente por James Martin en 1980, aunque anteriormente había sido utilizada por Barry Boehm y Scott Shultz en IBM. Finalmente fue formalizada por Martin en 1991 en el libro homónimo. Como su propio nombre indica, se trata de una metodología diseñada para el desarrollo rápido de aplicaciones.

Características generales

- Se trata de una metodología ágil, iterativa e incremental.
- Utiliza el prototipado como modelo de aproximación a la solución final.
- Se basa en el desarrollo iterativo, de manera que una nueva versión de *software* se generará en un ciclo cuya duración podrá oscilar entre una y quince jornadas.

- Encapsulamiento del tiempo: esta técnica consiste en dejar fuera del desarrollo determinadas funcionalidades ya identificadas en los requisitos, posponiéndola para iteraciones posteriores. De esta manera se consigue poder generar nuevas versiones en un plazo de tiempo relativamente corto.
- Grupos de proyecto reducidos: la utilización de esta metodología requiere grupos de trabajo pequeños, constituidos por gente motivada y muy versátil, capaz de desempeñar diferentes roles durante el desarrollo del proyecto.
- Basado en la utilización de herramientas: esta metodología recomienda encarecidamente la utilización de herramientas CASE para el diseño y la construcción del *software*. Los tipos de herramienta utilizadas por esta metodología son:

  – Integración de datos.
  – Entornos de desarrollo.
  – Herramientas de toma de requisitos.
  – Herramientas de modelado de datos.
  – Herramientas de generación de código.

Fases del proyecto

- Actividades preproyecto.

En esta fase se genera el Plan de Gestión del Proyecto. El mencionado plan debe generarse mediante una puesta en común de todos los intervinientes. Incluirá por lo menos los siguientes elementos:

– Los riesgos y sus estrategias de mitigación asociadas.
– Planificación inicial del desarrollo, incluyendo recursos, hitos y entregables.
– Estándares, herramientas y tecnologías a utilizar en el desarrollo del proyecto.
– Resultado esperado de la implantación del proyecto.
– Condiciones contractuales, incluyendo coste de recursos externos y/o herramientas.

- Planificación de requisitos.

Esta fase se conoce también como Etapa de definición conceptual. Consiste en una serie de reuniones entre el equipo encargado de la definición de requisitos y los usuarios finales. En estas reuniones se trabaja tanto en definir el alcance del proyecto como en elaborar una lista inicial de requisitos a implementar.

Los entregables de esta fase son:

– Lista de entidades a desarrollar.
– Diagramas de actividad que describan las interacciones entre dichas entidades.

La duración de esta fase idealmente debería comprender entre una y cuatro semanas. A la finalización de esta etapa, los requisitos deberían estar lo suficientemente maduros como para poder realizar una estimación basada en puntos función.

- Diseño de usuario.

En esta fase, también conocida como Etapa de diseño funcional, el equipo de analistas funcionales se reune con los usuarios finales en unas reuniones de trabajo denominadas JAD.

En estas sesiones de trabajo los analistas detallan más ampliamente los requisitos, transforman la planificación de requisitos de la fase anterior en un modelo de datos. Asimismo, se formalizan las reglas de negocio, se detallan los planes de prueba y se crean los flujos de pantallas esenciales del sistema. Una vez superado el ecuador de esta fase entra en juego el equipo de desarrollo denominado SWAT (Trabajadores Hábiles con Herramientas Avanzadas) para colaborar con los analistas en la creación del modelo de datos y en la identificación de los componentes reutilizables.

Antes de avanzar a la siguiente fase, los requisitos deberán introducirse en una aplicación adecuada y deberá realizarse una estimación de esfuerzo. Por otro lado, en esta fase deberá identificarse el contenido del primer prototipo ligero en funcionalidad a implementar. Este prototipo, aunque ligero, deberá implementar todos los requisitos considerados como esenciales. Con el objetivo de conseguir que dicha implementación no se dilate demasiado en el tiempo podrá omitirse el desarrollo de funcionalidades satélite dentro del prototipo, como podrían ser la validación de los datos introducidos en los formularios o las funcionalidades de subida y bajada de ficheros.

Por último, la duración de esta fase debería ser de entre tres y cinco semanas.

- Construcción.

Esta fase se divide en iteraciones con duración de entre una y diez jornadas de trabajo. Cada una de dichas iteraciones estará constituida por las siguientes actividades:

  – Codificación.
  – Pruebas.
  – Refinamiento de requisitos.

Las herramientas CASE seleccionadas deberán ser capaces de generar al menos el acceso a datos, aunque sería conveniente que incluyera la generación de lógica de negocio e interfaces de usuario. Utilizando herramientas adecuadas, el primer prototipo debería estar listo tras unas pocas jornadas de trabajo.

Una vez que el prototipo está concluido, éste se prueba utilizando los casos de prueba generados en la fase anterior. Primeramente, las pruebas serán realizadas por los analistas, a continuación, Negocio y, por último, se mantendrá una reunión de todos los intervinientes, los cuales constituyen el Grupo de enfoque, con el fin de determinar los requisitos de la siguiente iteración. La duración de estas reuniones debería ser de unas dos horas, para lo cual será necesario que el moderador mantenga el foco en aquellos temas más relevantes.

A la finalización de la reunión (o conjunto de reuniones), los analistas y desarrolladores deberán actualizar la documentación generada en la fase anterior:

- Requisitos.
- Modelo de datos.
- Plan de pruebas.
- Planificación del proyecto.

Cuando el prototipo alcance la madurez suficiente, las pruebas deberán realizarse teniendo en cuenta que su aprobación implicará el despliegue en producción de una nueva aplicación o subsistema. Para ello, deberá actualizarse adecuadamente la documentación y realizarse las pruebas de aceptación de usuario.

- Implantación.

Se denomina también Etapa de despliegue. En esta fase se desarrollan los interfaces con otros sistemas y se preparan las cargas de datos, como por ejemplo en el caso de las tablas tipo.

Además, en esta fase, se realiza la formación a los usuarios finales de la aplicación, aprovechando para ello la realización de las pruebas de aceptación.

Por último, cabe señalar que la duración de esta fase es dificil de determinar, ya que es muy dependiente de la tipología del proyecto.

- Actividades posimplementación.

Deberán realizarse al menos las siguientes tareas:

- Revisión de las métricas del proyecto.
- Inventario de los componentes reutilizables generados, el plan de proyecto y el plan de pruebas.
- Documento de lecciones aprendidas.

El siguiente esquema muestra el ciclo de vida de esta metodología:

Planificación del proyecto

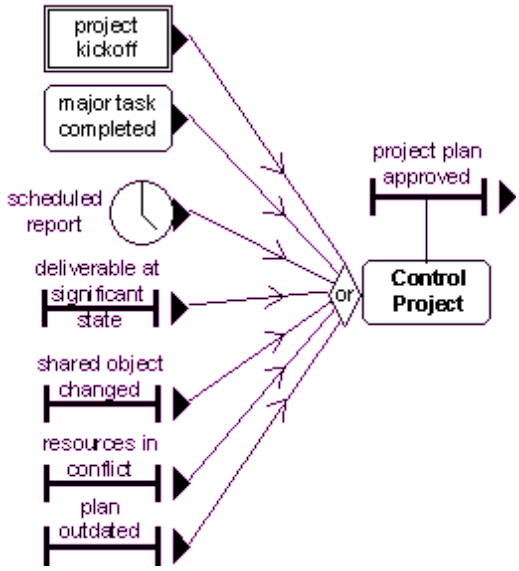
Tal y como hemos comentado anteriormente, para la planificación de un proyecto implementado con esta metodología es necesaria una primera planificación inicial del proyecto. Tras esta primera planificación podremos ir adaptando la planificación de cada una de las iteraciones, de manera que antes de abordar cada iteración podamos actualizar de manera realista el plan del proyecto.

Gestión del proyecto

La metodología incluye un conjunto de prácticas agrupadas dentro del denominado Control de Proyecto. Esto permite mantener el control del proyecto, ya que en ocasiones la naturaleza iterativa de esta metodología puede dar falsa sensación de avance. Siendo consistente con la metodología, este conjunto de prácticas trata de dejar a un lado la burocracia para permitir realizar el seguimiento del proyecto de la forma menos intrusiva posible.

La responsabilidad de este control recae directamente en el jefe de proyecto, el cual se encarga de realizar el seguimiento. En el caso de que el jefe de proyecto vea que las acciones correctoras no serán necesarias para corregir el rumbo del proyecto, será su responsabilidad hablar con el resto de intervinientes en el proyecto para modificar la planificación del mismo.

Existen una serie de eventos dentro del desarrollo del proyecto que llevan acarreada una ejecución del mecanismo Control de Proyecto. Pueden verse en el esquema adjunto:



Gestión de riesgos

- Riesgos funcionales: al tratarse de una metodología iterativa, cada iteración sirve como punto de revisión en el que puede observarse si la implementación se adapta a las necesidades funcionales de negocio, minimizando así este tipo de riesgo.
- Riesgos de plazo: los riesgos en plazo tienden a manifestarse de manera inherente con esta metodología. El proyecto se arranca con un conjunto limitado de requisitos que se irá completando conforme avance el proyecto, de manera que es muy dificil que la planificación inicial permita dar una fecha con el suficiente rigor.

3.4 **Hito 4.** Clasificación de las metodologías seleccionadas según las variables obtenidas en el hito 1.

Antes de evaluar las metodologías seleccionadas en función de las variables mencionadas, se incluye un cuadro resumen con sus características:

Metodología	Líneal	Iterativa	Ágil	Orientada a documentación	Adaptabilidad a cambio de requisitos	Foco
Ciclo de vida en cascada	SÍ	NO	NO	SÍ	NO	Control de errores
Ciclo de vida en espiral	SÍ	SÍ	NO	SÍ	NO	Adaptabilidad
Metodología basado en prototipos	NO	SÍ	NO	SÍ	SÍ	Requisitos adaptables
Metodología de entrega por etapas	SÍ	SÍ	NO	SÍ	NO	Reducción de riesgos
Programación extrema	NO	SÍ	SÍ	NO	SÍ	Reducción de plazos
Scrum	NO	SÍ	SÍ	NO	SÍ	Productividad
Crystal orange / web	SÍ	NO	SI	NO	SÍ	Personas
RAD	NO	SÍ	SI	NO	SÍ	Velocidad

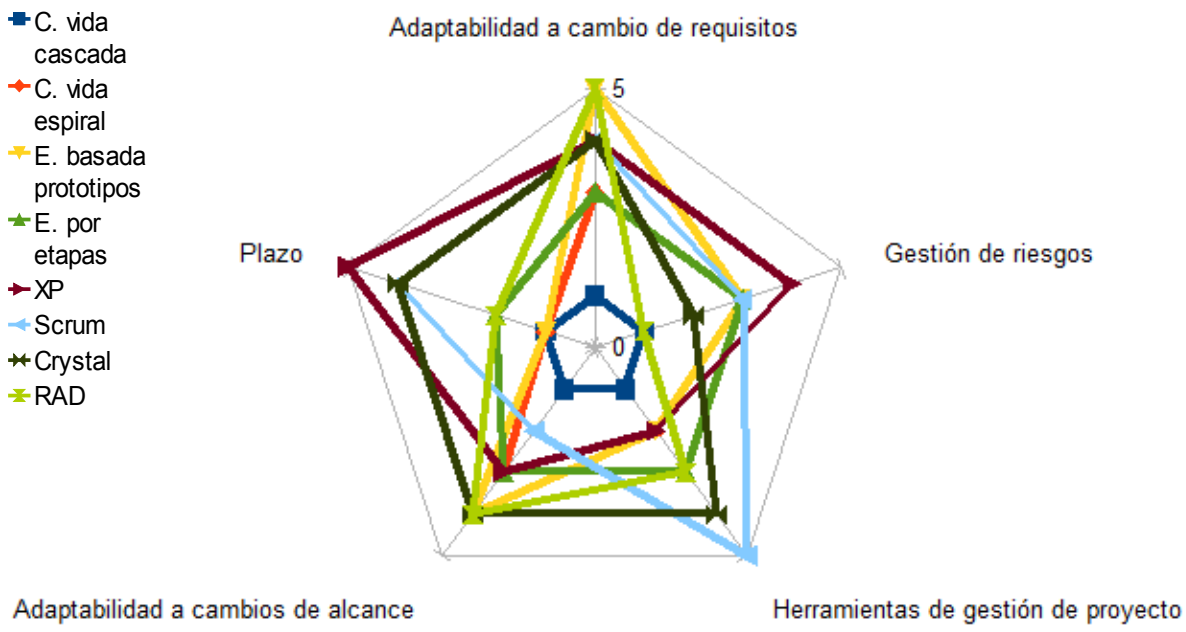
Recordamos los cinco parámetros seleccionados en el hito 1:

- Adaptabilidad a cambio de requisitos.
- Adaptabilidad a cambios de alcance.
- Plazo.
- Gestión de riesgos.
- Herramientas de gestión de proyecto.

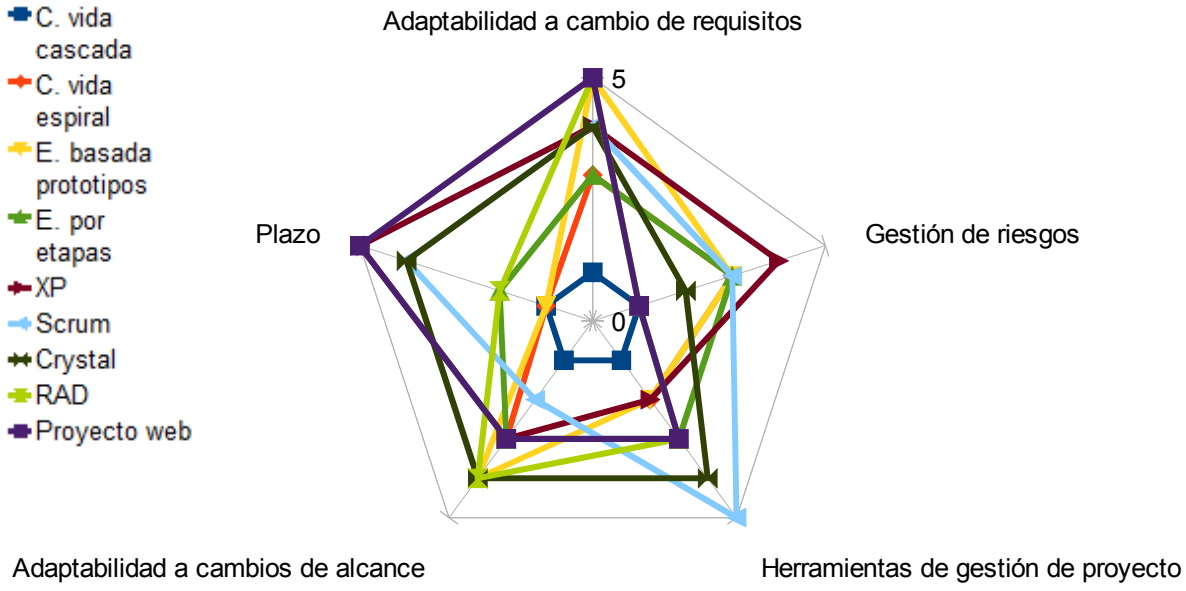
A continuación puede verse el cuadro resumen de puntuación para las ocho metodologías preseleccionadas:

Puntuación metodologías								
Metodología	**C. vida cascada**	**C. vida espiral**	**E. basada prototipos**	**E. por etapas**	**XP**	**Scrum**	**Crystal**	**RAD**
Adaptabilidad a cambio de requisitos	1	3	5	3	4	4	4	5
Adaptabilidad a cambios de alcance	1	3	4	3	3	2	4	4
Plazo	1	1	1	2	5	4	4	2
Gestión de riesgos	1	3	3	3	4	3	2	1
Herramientas de gestión de proyecto	1	2	2	3	2	5	4	3

Seguidamente podemos ver dichas puntuaciones expresadas en un gráfico:



A continuación se incluye en el gráfico los valores obtenidos para los proyectos *web* con el fin de contextualizar las ocho metodologías en el ámbito de los proyectos *web*:



4. Parte 2. Caso práctico.

4.1 **Hito 5:** Descripción de un proyecto estándar de desarrollo *web*.

En este apartado se dará una versión de alto nivel de los requisitos funcionales de un proyecto estándar de desarrollo *web* que servirá para poder realizar la división en tareas, estimación de esfuerzo y planificación inicial del mismo en apartados posteriores.

Una Entidad Aseguradora ha decidido renovar su portal w*eb* después de los resultados negativos de su encuesta de satisfacción a clientes/No clientes realizada hace unos meses.

En consecuencia, el cliente encarga la realización de un nuevo portal *web* para el Ramo del Automóvil. Los requisitos de la mencionada aplicación serán:

Código	**Descripción del requisito**
REQ-001	Tarificador de precios.
REQ-002	Comparar con otras Entidades Aseguradoras las pólizas estándar.
REQ-003	Geolocalización de talleres en un mapa.
REQ-004	Dar de alta en el club de fidelización de la Entidad Aseguradora.
REQ-005	Acceso al área privada donde el cliente dispondrá de la información de sus productos contratados, Club de fidelización y promociones.
REQ-006	Información sobre productos contratados (área privada).
REQ-007	Información del club de fidelización (área privada).
REQ-008	Información sobre promociones (área privada).

4.2 **Hito 6:** División en tareas y estimación de esfuerzo atendiendo a las metodologías seleccionadas.

4.2.1 Procedimiento de cálculo de esfuerzo.

Para que los datos obtenidos sean lo más ajustadas posibles, la realización de la estimación se realizará en cinco pasos:

a) División en tareas del proyecto: se generará una tabla con las tareas del proyecto.

b) Determinación del equipo del proyecto: atendiendo a la división en tareas se constituirá el equipo de trabajo adecuado para su realización.

c) Estimación utilizando criterio de experto: para cada una de las tareas se realiza una estimación basada en la experiencia en proyectos similares. Este método se basa en el juicio del experto. La salida de esta fase será la estimación de esfuerzo para cada una de las tareas detalladas.

d) División del esfuerzo entre los diferentes roles que participen en el proyecto.

e) Ajuste de la estimación utilizando el método del caso esperado: este método de estimación, tal y como nos relata Steve McConnell[1], consiste en encontrar, para cada una de las tareas, además del tiempo esperado, el tiempo en el mejor y en el peor de los casos. Posteriormente se aplicaría la siguiente fórmula:

$$Caso\ Esperado = \frac{[Mejor\ caso + (4 * Caso\ más\ probable) + Peor\ caso]}{6}$$

En cualquier caso, en nuestra estimación no se utilizará esta fórmula, sino la aproximación que utiliza Szuske, tratando de corregir las estimaciones demasiado optimistas:

$$Caso\ Esperado = \frac{[Mejor\ caso + (3 * Caso\ más\ probable) + (2 * Peor\ caso)]}{6}$$

El valor obtenido en la fórmula se ajustará siempre a un número entero.

Por último, cabe destacar que se descarta la obtención de una estimación basada en puntos función, ya que el proyecto seleccionado no está lo suficientemente definido como para realizar una estimación basada en funcionalidades. Recordemos que nos encontramos en una fase temprana del proyecto en la que solamente contamos con unos requisitos funcionales de alto nivel.

Salvo que se especifique lo contrario, para calcular el esfuerzo contaremos con un equipo de proyecto constituido por un jefe de proyectos, un arquitecto de *software*, un analista y cuatro programadores.

4.2.2 Ciclo de vida en cascada

A continuación podemos ver la división en tareas y la estimación de esfuerzo ajustada (entre paréntesis puede verse la estimación de esfuerzo mínima, normal y máxima de la tarea):

[1] McConnell, S. *Software Estimation*. Páginas 108-109

Descomposición en tareas		Personas / hora			
		JP	AN	AR	P
Análisis y diseño					
1	Análisis funcional		50(20,40,80)		
2	Diseño técnico		50(20,40,80)		
3	Diseño de los servicios *web*		50(20,40,80)		
Desarrollo					
1	Desarrollo de la estructura de aplicación			103(60,80,160)	
2	Desarrollo de las páginas				207(120,160,320)
3	Desarrollo de los servicios *web*				207(120,160,320)
4	Pruebas unitarias				93(80,80,120)
5	Correcciones tras las pruebas				100(40,80,160)
Pruebas					
1	Realización del plan de pruebas		93(80,80,120)		
2	Realización de las pruebas	73(40,80,80)			
Preimplantación					
1	Instalación y configuración de servidores			53(40,40,80)	
Implantación					
1	Paquetización aplicación			20(8,16,32)	
2	Despliegue aplicación en preproducción			20(8,16,32)	
3	Despliegue aplicación en producción			20(8,16,32)	
Gestión del proyecto					
1	Realización del documento de inicio del proyecto	8(8,8,8)			
2	Primera planificación del proyecto	8(8,8,8)			
3	Seguimiento y control del proyecto	53(40,40,80)			
4	Selección del equipo del proyecto	32(32,32,32)			
5	Reporte a otras áreas o comités	53(40,40,80)			
6	Realización del documento de fin del proyecto	8(8,8,8)			

4.2.3 Ciclo de vida en espiral

En este apartado nos centraremos en la implementación clásica del Ciclo de vida en espiral, que no es sino una versión iterada del Ciclo de vida en cascada. De esta manera, dado el tamaño relativamente pequeño de un proyecto *web*, se consideran suficientes dos iteraciones. La segunda iteración no incluirá las tareas de preimplantación, si bien las tareas de implantación en las dos iteraciones serán de igual duración que las realizadas en el Ciclo de vida en cascada.

A la hora de realizar la división en tareas se han tenido en cuenta las siguientes consideraciones:

- El desarrollo de la estructura de la aplicación se realizará integramente en la primera iteración.
- La instalación y configuración de servidores se realizará únicamente dentro de la primera iteración.

- El despliegue en producción se realizará únicamente en la segunda iteración, ya que la primera iteración sólo se subirá hasta el entorno de preproducción para que pueda ser vista por el usuario.

En la siguiente tabla puede verse una división en tareas del proyecto, con su correspondiente esfuerzo ajustado y estimado:

Descomposición en tareas		Personas / hora			
		JP	AN	AR	P
PRIMERA ITERACIÓN					
Análisis y diseño					
1	Análisis funcional		25(10,20,40)		
2	Diseño técnico		25(10,20,40)		
3	Diseño de los servicios *web*		25(10,20,40)		
Desarrollo					
1	Desarrollo de la estructura de la aplicación			103(60,80,160)	
2	Desarrollo de las páginas				103(60,80,160)
3	Desarrollo de los servicios *web*				103(60,80,160)
4	Pruebas unitarias				47(40,40,60)
5	Correcciones tras las pruebas				50(20,40,80)
Pruebas					
1	Realización del plan de pruebas		47(40,40,60)		
2	Realización de las pruebas	37(20,40,40)			
Preimplantación					
1	Instalación y configuración de servidores			53(40,40,80)	
Implantación					
1	Paquetización aplicación			20(8,16,32)	
2	Despliegue aplicación en preproducción			20(8,16,32)	
SEGUNDA ITERACIÓN					
Análisis y diseño					
1	Análisis funcional		25(10,20,40)		
2	Diseño técnico		25(10,20,40)		
3	Diseño de los servicios *web*		25(10,20,40)		
Desarrollo					
1	Desarrollo de las páginas				103(60,80,160)
2	Desarrollo de los servicios *web*				103(60,80,160)
3	Pruebas unitarias				47(40,40,60)
4	Correcciones tras las pruebas				50(20,40,80)
Pruebas					
1	Realización del plan de pruebas		47(40,40,60)		
2	Realización de las pruebas	37(20,40,40)			
Implantación					
1	Paquetización aplicación			20(8,16,32)	
2	Despliegue aplicación en preproducción			20(8,16,32)	

3	Despliegue aplicación en producción			20(8,16,32)	
Gestión del proyecto					
1	Realización del documento de inicio del proyecto	8(8,8,8)			
2	Primera planificación del proyecto	8(8,8,8)			
3	Seguimiento y control del proyecto	53(40,40,80)			
4	Selección del equipo del proyecto	32(32,32,32)			
5	Reporte a otras áreas o comités	53(40,40,80)			
6	Realización del documento de fin del proyecto	8(8,8,8)			

4.2.4 Metodología basada en prototipos

Antes de comenzar con la división en tareas del proyecto, estudiaremos una cuestión previa. Tal y como nos dice Steve McConnell[2], a la hora de afrontar un proyecto mediante la Metodología basada en prototipos, lo primero que nos tenemos que plantear es si el prototipo se va a utilizar única y exclusivamente para la generación de nuevos requisitos o bien si el prototipo formará parte del producto final.

En nuestro caso en concreto, resulta trivial decidir que el prototipo formará parte de la entrega final, ya que nos enfrentamos a un proyecto de migración en el que no se espera generar nuevos requisitos funcionales sino construir una aplicación.

Ante esta decisión será necesario realizar algún ajuste en la composición del equipo. Lo primero que se pone de manifiesto es que el diseño de la arquitectura que se realice en el prototipo será una versión reducida de la arquitectura de la versión final, lo cual hace que para afrontar la fase de prototipado sea necesario doblar el equipo de arquitectos previsto en el Ciclo de vida en cascada.

Asimismo, la permanencia de los arquitectos en el equipo no se hará necesaria desde la finalización del prototipo hasta que la aplicación esté lista para ser desplegada entre entornos, ya que una vez finalizado el prototipo la arquitectura estará lo suficientemente afinada como para que los desarrolladores puedan encargarse de hacerla crecer hasta la versión final.

Se estima que la realización de las tareas propias del prototipo conlleve un esfuerzo equivalente a un quinto del esfuerzo planificado para las tareas análogas en el Ciclo de vida en cascada, a excepción de las siguientes actividades:

- Desarrollo de la infraestructura de la aplicación: será la mitad de lo estimado.
- Tareas de despliegue del prototipo: el esfuerzo de despliegue del prototipo será el mismo que el esfuerzo de despliegue de la aplicación completa.
- Tareas de gestión de proyecto: se mantienen estables. A pesar de que la gestión durante la fase de prototipado aumenta, se considera que disminuye durante la fase de desarrollo de la aplicación definitiva, de manera que llegan a converger.

Por otro lado, la tarea de preimplantación se desplaza para ser realizada durante la fase de prototipado.

[2] McConnell, S. Rapid Development. Páginas 433 - 443

En la siguiente tabla puede verse la división en tareas y la estimación, tanto una vez ajustada como estimada (entre paréntesis):

Descomposición en tareas		Personas / hora			
		JP	AN	AR	P
Análisis, diseño y desarrollo del prototipo					
1	Análisis funcional		10(4,8,16)		
2	Diseño técnico		10(4,8,16)		
3	Diseño de los servicios *web*		10(4,8,16)		
4	Desarrollo de la infraestructura del prototipo			52(30,40,80)	
5	Desarrollo del prototipo				83(48,64,128)
6	Pruebas y correcciones tras las pruebas				39(24,32,56)
Pruebas de aceptación del prototipo					
1	Realización del plan de pruebas		19(16,16,24)		
2	Realización de las pruebas	15(8,16,16)			
Preimplantación del prototipo					
1	Instalación y configuración de servidores			53(40,40,80)	
Implantación del prototipo					
1	Paquetización del prototipo			20(8,16,32)	
2	Despliegue del prototipo en preproducción			20(8,16,32)	
3	Despliegue del prototipo en producción			20(8,16,32)	
Análisis y diseño					
1	Análisis funcional		40(16,32,64)		
2	Diseño técnico		40(16,32,64)		
3	Diseño de los servicios *web*		40(16,32,64)		
Desarrollo					
1	Desarrollo de la estructura de la aplicación			52(30,40,80)	
2	Desarrollo de las páginas				165(96,128,256)
3	Desarrollo de los servicios *web*				165(96,128,256)
4	Pruebas unitarias				75(64,64,96)
5	Correcciones tras las pruebas				80(32,64,128)
Pruebas					
1	Realización del plan de pruebas		75(64,64,96)		
2	Realización de las pruebas	59(32,64,64)			
Implantación					
1	Paquetización aplicación			20(8,16,32)	
2	Despliegue aplicación en preproducción			20(8,16,32)	
3	Despliegue aplicación en producción			20(8,16,32)	
Gestión del proyecto					
1	Realización del documento de inicio del proyecto	8(8,8,8)			
2	Primera planificación del proyecto	8(8,8,8)			
3	Seguimiento y control del proyecto	53(40,40,80)			

4	Selección del equipo del proyecto	32(32,32,32)			
5	Reporte a otras áreas o comités	53(40,40,80)			
6	Realización del documento de fin del proyecto	8(8,8,8)			

Para realizar el ajuste se han tenido en cuenta las siguientes consideraciones:

- El peor caso posible de aquellas tareas que hayan sido realizadas en el prototipo se verá reducido. Así las cosas, consideramos que el plazo esperado y el peor caso coinciden para aquellas tareas de desarrollo y diseño que se realizan con posterioridad al piloto.
- En el resto de los casos, tanto el mejor caso como el peor caso serán proporcionales a los obtenidos para el Ciclo de vida en cascada.

4.2.5 Metodología de entrega por etapas

Atendiendo a esta metodología, deberemos dividir el proyecto completo en una serie de etapas tales que los hitos de entrega de las mismas sean lo suficientemente cercanos como para generar al menos un entregable semanal. Cada una de esas entregas irá acompañada de su correspondiente despliegue de manera que la funcionalidad desarrollada quede disponible para su utilización por parte de los usuarios finales.

Además, debemos tener en cuenta que la arquitectura de la solución deberá desplegarse con la primera de las fases que se despliegue en el entorno de producción.

En la siguiente tabla puede verse la división en tareas, su esfuerzo (tanto ajustado como estimado) y el trabajo de los diferentes miembros del equipo, que será análogo al constituido en el Ciclo de vida en cascada:

Descomposición en tareas		Personas / hora			
		JP	AN	AR	P
FASE I					
Desarrollo arquitectura					
1	Desarrollo de la estructura de aplicación			103(60,80,160)	
Preimplantación aplicación					
1	Instalación y configuración de servidores			53(40,40,80)	
Análisis y diseño fase I					
1	Análisis funcional		17(7,13,27)		
2	Diseño técnico		17(7,13,27)		
3	Diseño de los servicios *web*		17(7,13,27)		
Desarrollo fase I					
1	Desarrollo de las páginas				69(40,53,107)
2	Desarrollo de los servicios *web*				69(40,53,107)
3	Pruebas unitarias				31(27,27,40)
4	Correcciones tras las pruebas				33(13,27,53)

Pruebas fase I					
1	Realización del plan de pruebas		31(27,27,40)		
2	Realización de las pruebas	25(13,27,27)			
Implantación fase I					
1	Paquetización aplicación			20(8,16,32)	
2	Despliegue aplicación en preproducción			20(8,16,32)	
3	Despliegue aplicación en producción			20(8,16,32)	
FASE II					
Análisis y diseño fase II					
1	Análisis funcional		17(7,13,27)		
2	Diseño técnico		17(7,13,27)		
3	Diseño de los servicios *web*		17(7,13,27)		
Desarrollo fase II					
1	Desarrollo de las páginas				69(40,53,107)
2	Desarrollo de los servicios *web*				69(40,53,107)
3	Pruebas unitarias				31(27,27,40)
4	Correcciones tras las pruebas				33(13,27,53)
Pruebas fase II					
1	Realización del plan de pruebas		31(27,27,40)		
2	Realización de las pruebas	25(13,27,27)			
Implantación fase II					
1	Paquetización aplicación			20(8,16,32)	
2	Despliegue aplicación en preproducción			20(8,16,32)	
3	Despliegue aplicación en producción			20(8,16,32)	
FASE III					
Análisis y diseño fase III					
1	Análisis funcional		17(7,13,27)		
2	Diseño técnico		17(7,13,27)		
3	Diseño de los servicios *web*		17(7,13,27)		
Desarrollo fase III					
1	Desarrollo de las páginas				69(40,53,107)
2	Desarrollo de los servicios *web*				69(40,53,107)
3	Pruebas unitarias				31(27,27,40)
4	Correcciones tras las pruebas				33(13,27,53)
Pruebas fase III					
1	Realización del plan de pruebas		31(27,27,40)		
2	Realización de las pruebas	25(13,27,27)			
Implantación fase III					
1	Paquetización aplicación			20(8,16,32)	
2	Despliegue aplicación en preproducción			20(8,16,32)	
3	Despliegue aplicación en producción			20(8,16,32)	
Gestión del proyecto					
1	Realización del documento de inicio del proyecto	8(8,8,8)			

2	Primera planificación del proyecto	8(8,8,8)			
3	Seguimiento y control del proyecto	53(40,40,80)			
4	Selección del equipo del proyecto	32(32,32,32)			
5	Reporte a otras áreas o comités	53(40,40,80)			
6	Realización del documento de fin del proyecto	8(8,8,8)			

Para realizar el ajuste hemos tenido en cuenta lo obtenido para el Ciclo de vida en cascada de manera proporcional a las tres fases en las que se ha dividido el desarrollo (40%, 40%, 20%), dejando al margen las tareas de despliegue que deben realizarse de manera completa en cada una de las fases.

4.2.6 Programación extrema (XP)

Para poder realizar la división en tareas es necesario dividir el equipo en dos equipos de desarrollo diferenciados. Uno de ellos estará formado por un arquitecto y dos programadores y el otro por un analista y dos programadores.

Por otro lado, el jefe de proyecto se encargará de la preparación del plan de pruebas. Asimismo, se encargará de la realización de las pruebas, si bien estas se realizarán dentro del proceso de pruebas unitarias del equipo de desarrollo y con la presencia de un usuario final de la aplicación que tenga potestad suficiente para dar por válidas dichas pruebas.

Con ello, se estima que el tiempo de pruebas unitarias aumente en un 25%, pero que desaparezca el tiempo de pruebas de aceptación. Asimismo, desaparece la incertidumbre en la realización de dichas pruebas. Por otro lado, se considera una jornada adicional de documentación al finalizar cada versión. Las realización de esta documentación correrá a cargo del jefe de proyecto.

A continuación puede verse la división en tareas y sus estimaciones asociadas, tanto ajustadas como estimadas.

Descomposición en tareas		Personas/hora		
		EQ1	EQ2	JP
Análisis y diseño, desarrollo y pruebas				
1	Análisis, diseño y desarrollo primera versión	117(47,93,187)	117(47,93,187)	
2	Pruebas primera versión			33(33,33,33)
3	Documentación primera versión			8(8,8,8)
4	Análisis, diseño y desarrollo segunda versión	117(47,93,187)	117(47,93,187)	
5	Pruebas segunda versión			33(33,33,33)
6	Documentación segunda versión			8(8,8,8)
7	Análisis, diseño y desarrollo tercera versión	117(47,93,187)	117(47,93,187)	
8	Pruebas tercera versión			33(33,33,33)
9	Documentación tercera versión			8(8,8,8)
Preimplantación e implantación				
1	Instalación y configuración de servidores	53(40,40,80)		

2	Paquetización de la aplicación		20(8,16,32)	
3	Despliegue aplicación en preproducción	20(8,16,32)		
4	Despliegue aplicación en producción		20(8,16,32)	
Gestión del proyecto				
1	Realización del documento de inicio del proyecto			8(8,8,8)
2	Primera planificación del proyecto			8(8,8,8)
3	Seguimiento y control del proyecto			53(40,40,80)
4	Selección del equipo del proyecto			32(32,32,32)
5	Reporte a otras áreas o comités			53(40,40,80)
6	Realización del documento de fin del proyecto			8(8,8,8)

4.2.7 Scrum

Para poder realizar una estimación de esfuerzo con esta metodología, lo primero que debemos hacer es determinar la duración de las distintos *Sprints* de los que va a constar el trabajo.

Contando con el equipo de trabajo estándar (un jefe de proyecto al que denominaremos Scrum master, un arquitecto de *software*, un analista y cuatro programadores) y el *backlog* o pila de producto a desarrollar, se decide que los *Sprint* sean de un mes.

Para la determinación de los plazos de las tareras partiremos de los tiempos estimados para el Ciclo de vida en cascada, teniendo en cuenta las siguientes consideraciones:

- Las tareas de gestión se realizan en las reuniones pertinentes que se realizan durante cada uno de los *Sprint*.
- El concepto de revisión de requisitos no existe como tal, sino que se trata en las reuniones pertinentes que tienen lugar dentro de cada *Sprint*.
- El documento de inicio y cierre del proyecto tampoco existen como tal, ya que en las reuniones de cierre de cada *Sprint* se da la información pertinente a los usuarios.
- Las tareas de despliegue deben realizarse de manera íntegra en cada uno de los *Sprints*.
- El análisis funcional de cada uno de los *Sprints* se desarrolla en el prepartido, y se denomina diseño de arquitectura de alto nivel.
- La paquetización o refactorización de la aplicación no se considera una tarea en sí, ya que el equipo de trabajo realiza sus tareas como una melé y no es necesario realizar una integración compleja entre el trabajo de los diferentes miembros del equipo.
- A la hora de realizar la planificación habrá que tener en cuenta que el equipo de trabajo no se incorpora hasta que no esté completado el prepartido del primero de los *Sprint*.

Descomposición en tareas	Personas/hora	
	Equipo	Scrum master
PRIMER SPRINT		

Prepartido			
1	Planificación del *Sprint*		8(8,8,8)
2	Diseño de la arquitectura de alto nivel		13(7,13,27)
Partido (*backlog* de tareas a realizar)			
1	Desarrollo de servicios *web* de acceso y configuración escritorio	100(40,80,160)	
2	Desarrollo de las páginas de acceso y configuración escritorio	100(40,80,160)	
3	Pruebas de acceso y configuración de escritorio	64(40,53,93)	
4	Instalación y configuración de servidores	53(40,40,80)	
5	Despliegue en preproducción del *Sprint*	20(8,16,32)	
Post-partido (cierre del *Sprint*)			
1	Pruebas de aceptación del *Sprint*		55(40,53,66)
2	Despliegue en producción del *Sprint*	20(8,16,32)	
3	Documentación del *Sprint*		36(27,27,53)
4	Reunión de revisión del *Sprint*	12(12,12,12)	2(2,2,2)
SEGUNDO SPRINT			
Prepartido			
1	Planificación del *Sprint*		8(8,8,8)
2	Diseño de la arquitectura de alto nivel		13(7,13,27)
Partido (*backlog* de tareas a realizar)			
1	Desarrollo de servicios *web* de la parte pública	100(40,80,160)	
2	Desarrollo de las páginas de la parte pública	100(40,80,160)	
3	Pruebas de la parte pública	64(40,53,93)	
4	Despliegue en preproducción del *Sprint*	20(8,16,32)	
Post-partido (cierre del *Sprint*)			
1	Pruebas de aceptación		110(80,106,132)
2	Despliegue en producción del *Sprint*	20(8,16,32)	
3	Documentación del *Sprint*		72(54,54,106)
4	Reunión de revisión del *Sprint*	12(12,12,12)	2(2,2,2)
TERCER SPRINT			
Prepartido			
1	Planificación del *Sprint*		8(8,8,8)
2	Diseño de la arquitectura de alto nivel		13(7,13,27)
Partido (*backlog* de tareas a realizar)			
1	Desarrollo de servicios *web* de la parte privada	100(40,80,160)	
2	Desarrollo de las páginas de la parte privada	100(40,80,160)	
3	Pruebas de la parte privada	64(40,53,93)	
4	Despliegue en preproducción del *Sprint*	20(8,16,32)	
Post-partido (cierre del *Sprint*)			
1	Pruebas de aceptación		110(80,106,132)
2	Despliegue en producción del *Sprint*	20(8,16,32)	
3	Documentación del *Sprint*		72(54,54,106)
4	Reunión de revisión del *Sprint*	12(12,12,12)	2(2,2,2)

4.2.8 Crystal orange / web

En el siguiente cuadro se encuentra la división en tareas con esta metodología. El equipo de proyecto estará constituido por dos analista de negocio, un arquitecto y cuatro programadores.

Hay que tener en cuenta las siguientes puntualizaciones:

- El plan de pruebas como tal no existe, ya que en cada uno de los ciclos los analistas de negocio validan las pruebas del grupo de desarrollo en la reunión de cierre del ciclo.
- El mismo hecho del punto anterior hace que las pruebas de aceptación se reduzcan en gran parte. Se estima que esa reducción sea de un 50% del tiempo estimado para el ciclo de vida en cascada.
- Los analistas de negocio deberán estar presentes durante la realización de las pruebas, no así durante las correcciones implícitas en las pruebas. Debido a que el número de analistas es la mitad que el de programadores, se estima que su esfuerzo será equivalente al de éstos.
- La reunión de cierre del ciclo se estima en una jornada para cada miembro del equipo de desarrollo.
- El análisis funcional del proyecto incluye unos entregables muy concretos que permiten definir con gran nivel de detalle el trabajo a realizar por los desarrolladores. Si bien esto no reduce el tiempo de desarrollo estándar, consideramos que elimina la incertidumbre al alza en las actividades de desarrollo, con lo que los plazos estimados se reducen.
- A la hora de realizar la planificación, tendremos en cuenta que el equipo de desarrollo no se incorpora al proyecto hasta que el analista de negocio (recordemos que en esta metodología el analista funcional y el analista técnico confluyen en este rol) ha realizado gran parte del diseño de la aplicación.

Descomposición en tareas		Personas / hora		
		AN	AR	P
Definición				
1	Redacción caso de uso negocio y caso de uso sistema	80(80,80,160)		
2	Definición del flujo básico de la aplicación		52(30,40,80)	
3	Diseño de las pantallas		25(10,20,40)	
4	Redacción de casos de uso	80(80,80,160)		
5	Diseño flujos de datos		25(10,20,40)	
Primer ciclo desarrollo				
1	Diseño de los servicios *web*		50(20,40,80)	
2	Desarrollo de la estructura de la aplicación		52(30,40,80)	
2	Desarrollo de los servicio *web*			51(40,53,53)
3	Desarrollo de las pantallas			51(40,53,53)
4	Pruebas y correcciones tras las pruebas	51(40,53,53)		51(40,53,53)
5	Reunión de cierre del ciclo	16(16,16,16)	8(8,8,8)	32(32,32,32)
Segundo ciclo desarrollo				
1	Diseño de los servicios *web*			
2	Desarrollo de los servicios *web*			51(40,53,53)
3	Desarrollo de las pantallas			51(40,53,53)
4	Pruebas y correcciones tras las pruebas	51(40,53,53)		51(40,53,53)
5	Reunión de cierre del ciclo	16(16,16,16)	8(8,8,8)	32(32,32,32)

Tercer ciclo desarrollo				
1	Diseño de los servicios *web*			
2	Desarrollo de los servicios *web*			51(40,53,53)
3	Desarrollo de las pantallas			51(40,53,53)
4	Pruebas y correcciones tras las pruebas	51(40,53,53)		51(40,53,53)
5	Reunión de cierre del ciclo	16(16,16,16)	8(8,8,8)	32(32,32,32)
Pruebas de aceptación				
1	Realización de las pruebas	37(20,40,40)		
Implantación				
1	Instalación y configuración de servidores		53(40,40,80)	
2	Paquetización aplicación		20(8,16,32)	
3	Despliegue aplicación en preproducción		20(8,16,32)	
4	Despliegue aplicación en producción		20(8,16,32)	
Gestión del proyecto				
1	Realización del documento de inicio del proyecto	8(8,8,8)		
2	Primera planificación del proyecto	8(8,8,8)		
3	Seguimiento y control del proyecto	53(40,40,80)		
4	Selección del equipo del proyecto	32(32,32,32)		
5	Reporte a otras áreas o comités	53(40,40,80)		
6	Realización del documento de fin del proyecto	8(8,8,8)		

4.2.9 Rapid Application Development (RAD)

A continuación podemos ver la división en tareas con esta metodología. En este caso no distinguiremos diferentes roles en la realización de cada una de las tareas, ya que esta metodología está basada en la versatilidad y capacidad del equipo, así como en el uso de herramientas específicas para la realización de cada una de las tareas. El tamaño del equipo será de siete miembros.

Para determinar los plazos de cada una de las tareas se han tenido en cuenta las siguientes consideraciones:

- Para la selección de las herramientas se estima una jornada de todo el equipo.
- La fase de JAD equivaldrá al análisis funcional más la primera estimación del proyecto.
- La fase de refinamiento de requisitos tendrá el esfuerzo de una jornada para cada uno de los siete miembros del equipo de desarrollo.

En la siguiente tabla puede verse tanto la división en tareas como la estimación de éstas (estimada y ajustada)

Descomposición en tareas		Personas/hora
Actividades preproyecto		
1	Selección de herramientas	56(56,56,56)
Diseño de usuario		

1	Refinamiento de requisitos (JAD)	58(28,48,88)
2	Análisis, diseño, desarrollo y pruebas prototipo ligero	103(60,80,160)
Construcción (primera iteración)		
1	Análisis, diseño y desarrollo	236(133,187,360)
2	Pruebas	56(40,53,67)
3	Paquetización aplicación	20(8,16,32)
4	Despliegue aplicación en preproducción	20(8,16,32)
5	Despliegue aplicación en producción	20(8,16,32)
6	Refinamiento de requisitos	56(56,56,56)
Construcción (segunda iteración)		
1	Análisis, diseño y desarrollo	236(133,187,360)
2	Pruebas	56(40,53,67)
3	Paquetización aplicación	20(8,16,32)
4	Despliegue aplicación en preproducción	20(8,16,32)
5	Despliegue aplicación en producción	20(8,16,32)
6	Refinamiento de requisitos	56(56,56,56)
Construcción (tercera iteración)		
1	Análisis, diseño y desarrollo	236(133,187,360)
2	Pruebas	56(40,53,67)
3	Paquetización aplicación	20(8,16,32)
4	Despliegue aplicación en preproducción	20(8,16,32)
5	Despliegue aplicación en producción	20(8,16,32)
6	Refinamiento de requisitos	56(56,56,56)
Implantación		
1	Instalación y configuración servidores	53(40,40,80)
2	Paquetización aplicación	20(8,16,32)
3	Despliegue aplicación en preproducción	20(8,16,32)
4	Despliegue aplicación en producción	20(8,16,32)
Gestión del proyecto		
1	Realización del documento de inicio del proyecto	8(8,8,8)
2	Primera planificación del proyecto	8(8,8,8)
3	Seguimiento y control del proyecto	53(40,40,80)
4	Selección del equipo del proyecto	32(32,32,32)
5	Reporte a otras áreas o comités	53(40,40,80)
6	Realización del documento de fin del proyecto	8(8,8,8)

4.3 **Hito 7:** Identificación de los hitos de seguimiento que permiten evaluar el grado de éxito de un proyecto *web*.

Uno de los aspectos más importantes a la hora de realizar el seguimiento de un proyecto es el concepto de valor ganado (earned value). Este concepto hace referencia a aquellos hitos de seguimiento cuya consecución representa ganancia de valor en un proyecto, de manera que la consecución de todos los hitos que aportan valor ganado representan un 100% de la implantación del proyecto.

Así las cosas, seleccionamos una serie de hitos que representarán la ganancia de valor de nuestro proyecto *web*:

- INI_[metodología]: indica la fecha de inicio del proyecto.
- FIN_REQ_[metodología]: indica la última fecha en la que los peticionarios del proyecto podrán dar de alta nuevos requistos.
- INI_DES_[metodología]: indica la fecha en la que el desarrollo comienza.
- INI_PRE_[metodología]: indica la fecha en la que las pruebas de aceptación comienzan.
- INI_PRO_[metodología]: indica la fecha en la que la primera funcionalidad está disponible en el entorno de producción.
- FIN_[metodología]: indica la fecha de fin del proyecto.

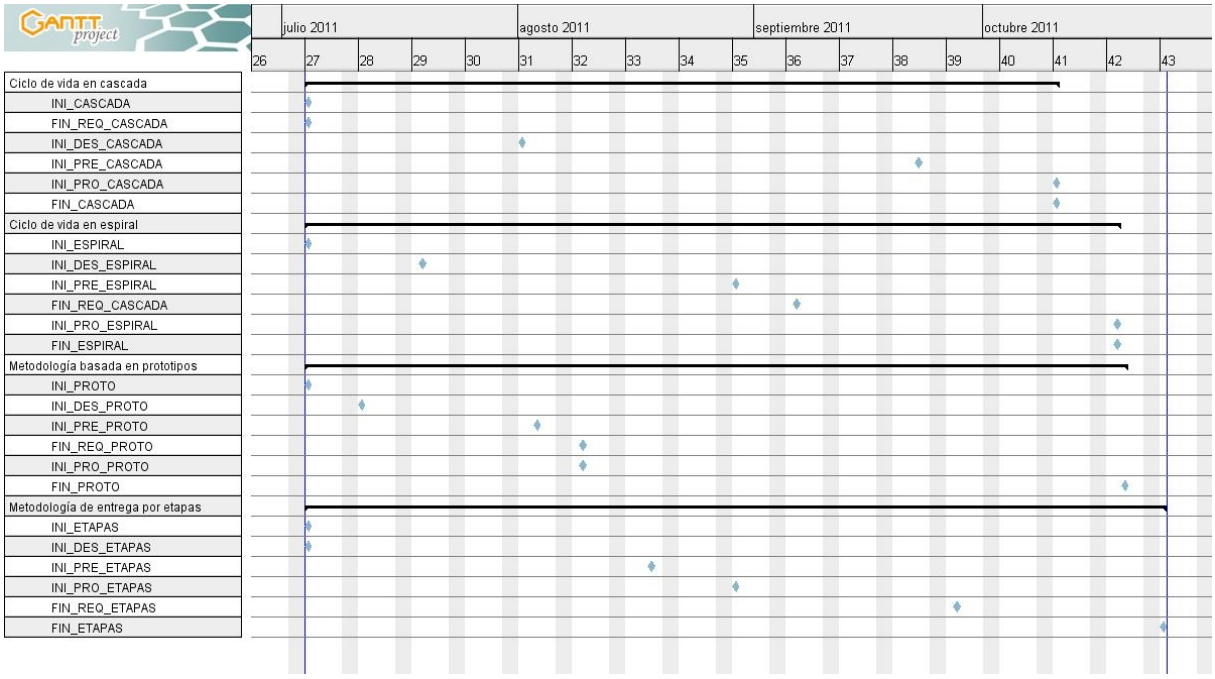
La codificación de las metodología en estos hitos estándar será la siguiente:

Codificación metodologías	
Metodología	Codificación
Ciclo de vida en cascada	CASCADA
Ciclo de vida en espiral	ESPIRAL
Metodología basada en prototipos	PROTO
Metodología de entrega por etapas	ETAPAS
Extreme programming (XP)	XP
Scrum	SCRUM
Crystal orange / web	CRYSTAL
Rapid Application Development (RAD)	RAD

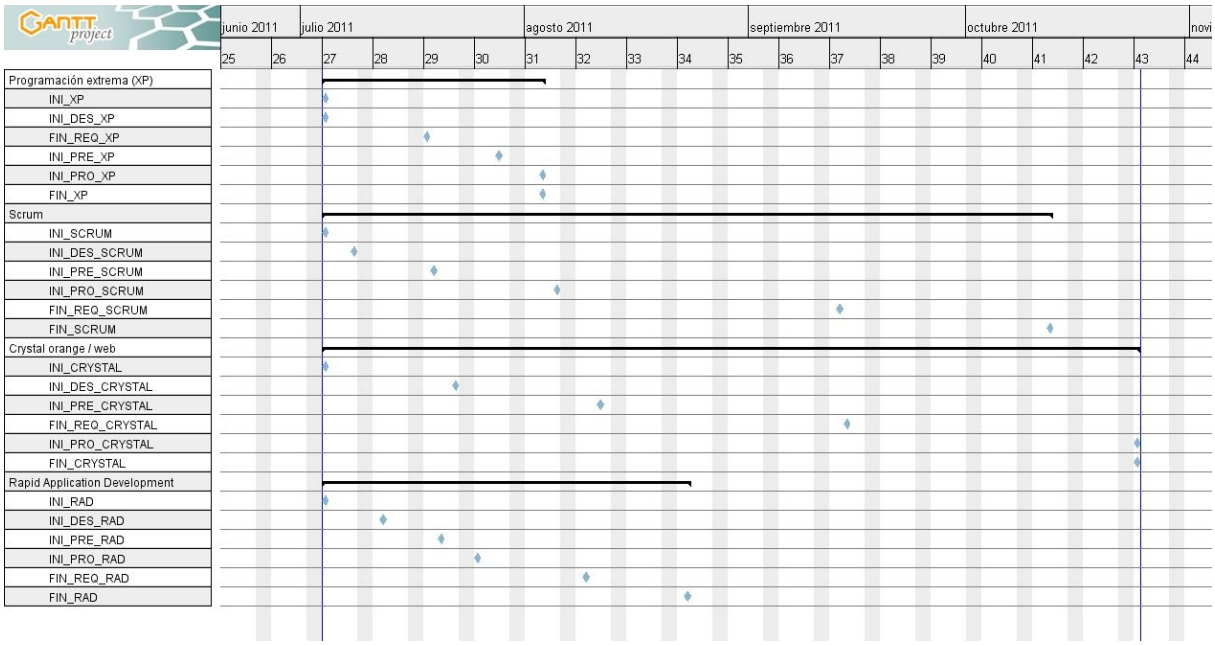
4.4 **Hito 8:** Planificación atendiendo a las metodologías seleccionadas.

En este apartado veremos la planificación de los proyectos atendiendo a las ocho metodologías seleccionadas. Para poder realizar una comparación lo más objetiva posible, tendremos en cuenta los hitos estándar descritos en el hito 7. A la hora de obtener el diagrama de Gannt, consideraremos que todos los proyectos comienzan el 4 de julio de 2011.

4.4.1 Metodologías clásicas



4.4.2 Metodologías ágiles



5. Parte 3. Conclusiones.

5.1 Conclusiones del estudio analítico.

Código	Conclusión
C1.1	Las metodologías ágiles son más adaptables a los cambios de requisitos que las metodologías clásicas.
C1.2	La Metodología basada en prototipos se adapta a los cambios de requisitos tan bien como la mejor metodología ágil (RAD).
C1.3	La Metodología basada en prototipos y RAD obtienen la misma puntuación en adaptabilidad a cambio de requisitos que la importancia otorgada a esta variable para los proyectos de desarrollo *web*.
C1.4	Las metodologías ágiles se adaptan mejor a los cambios de alcance que las metodologías clásicas.
C1.5	Scrum se adapta peor a los cambios de alcance que las metodologías clásicas iterativas (Ciclo de vida en espiral, Metodología basada en prototipos y Metodología de entrega por etapas).
C1.6	Aunque ninguna metodología se adapta tan bien a los cambios de alcance como requieren los proyectos de desarrollo en entorno *web*, las metodologías que más se aproximan son Metodología basada en prototipos, Crystal orange / web y RAD.
C1.7	Las metodologías ágiles consiguen menores tiempos de implantación que las metodologías clásicas.
C1.8	Ninguna metodología clásica alcanza el nivel requerido en cuanto a plazo de implantación en proyectos de desarrollo *web.*
C1.9	Scrum, Crystal orange / web y RAD obtienen mejores puntuaciones que las requeridas para adaptarse al desarrollo *web* en lo relativo a plazo, superándola en 1 punto (Crystal orange / web y RAD) y 2 puntos (Scrum) respectivamente.
C1.10	En lo relativo a la gestión de riesgos no es posible discernir qué tipo de metodología se adapta mejor, ya que obtienen puntuaciones muy similares.
C1.11	Cuatro de las metodologías alcanzan el umbral requerido en gestión de riesgos para los proyectos *web*. Son tres metodologías clásicas (Ciclo de vida en espiral, Metodología basada en prototipos y Metodología de entrega por etapas) y una ágil (Scrum).
C1.12	Programación extrema (XP) obtiene 4 puntos en gestión de riesgos, superando el umbral requerido para los proyectos de desarrollo *web*.
C1.13	Las metodologías ágiles ofrecen mejores herramientas de gestión de proyecto que las metodologías clásicas.
C1.14	Todas las metodologías objeto de estudio alcanzan el umbral requerido por los proyectos de desarrollo *web* en lo que a herramientas de gestión de proyecto se refiere.
C1.15	Teniendo en cuenta las conclusiones del estudio analítico, las metodologías ágiles se adaptan mejor a los proyectos *web* que las metodologías clásicas.
C1.16	Teniendo en cuenta las conclusiones del estudio analítico, la metodología que mejor se adapta a los proyectos *web* es la Metodología basada en prototipos y Crystal orange / web (3 puntos globales para alcanzar el umbral) seguida de RAD (4 puntos globales para alcanzar el umbral).

5.2 Conclusiones del caso práctico.

Código	Conclusión
C2.1	Las metodologías ágiles inician antes el desarrollo (hito INI_DES_[metodología]) que las metodologías clásicas, a excepción de Metodología de entrega por etapas, que inicia el desarrollo de la arquitectura la primera jornada.
C2.2	XP y Scrum inician el desarrollo la primera semana de trabajo, RAD y metodología basada en prototipos la segunda y Scrum la tercera.

C2.3	Ciclo de vida en cascada es la metodología que más tarde inicia los desarrollos.
C2.4	La segunda metodología que más tarde inicia el desarrollo es Crystal, debido fundamentalmente al tiempo necesario para seleccionar las herramientas.
C2.5	Las metodologías ágiles son capaces de entregar *software* a probar antes que las metodologías clásicas.
C2.6	La metodología basada en prototipos consigue entregables en preproducción antes que una de las cuatro metodologías ágiles, XP.
C2.7	Las metodologías ágiles se adaptan mejor a los cambios de requisitos, ya que los requisitos pueden ser modificados sin impactar al trabajo ya realizado hasta poco tiempo antes de la finalización del proyecto (distancia entre el hito FIN_REQ_[metodología] y FIN_[metodología]). Los requisitos pueden ser modificados 2 semanas antes de la finalización en RAD, 3 semanas antes en XP, 4 semanas antes en Scrum (coincidiendo con la duración de un Sprint) y 6 semanas en Crystal orange / web.
C2.8	Dos metodologías clásicas consiguen una adaptabilidad a los cambios de requisitos similar a la de las metodologías ágiles. Ciclo de vida en espiral permite la modificación de requisitos sin impacto en el trabajo realizado 6 semanas antes de la finalización, siendo este plazo de 4 semanas en el caso de la Metodología de entrega por etapas.
C2.9	Las metodologías ágiles consiguen tener antes parte del proyecto en producción, aunque en este aspecto los resultados son más ajustados.
C2.10	La metodología que antes consigue tener un entregable en producción es RAD (4ª semana), seguida de Scrum (5ª semana) y Metodología basada en prototipos (6ª semana). No demasiado lejos encontramos a Programación extrema (8ª semana) y Metodología de entrega por etapas (9ª semana).
C2.11	Las metodologías ágiles consiguen por lo general un plazo de implantación menor que las metodologías clásicas.
C2.12	Programación extrema (XP) y RAD finalizan el proyecto en 8 semanas de trabajo. El resto de metodologías finaliza tras un plazo entre 15 y 17 semanas de trabajo.

5.3 Conclusiones generales.

Código	Conclusión
C3.1	**Según el presente TFC, las metodologías ágiles se adaptan mejor a los proyectos de desarrollo *web* que las metodologías clásicas, ya que tanto en el estudio analítico como en el caso práctico hemos visto que superan en los cinco parámetros especificados a las metodologías clásicas.**
C3.2	Algunas metodologías clásicas (metodología basada en prototipos y metodología de entrega por etapas) consiguen un buen grado de adaptación a los proyectos *web*, similar al de las metodologías clásicas.
C3.3	Según el estudio analítico, las metodologías mejor adaptadas a los proyectos *web* son Crystal orange / web y Metodología basada en prototipos, seguidas de RAD.
C3.4	Según el caso práctico, la metodología mejor adaptada a proyectos *web* es RAD, seguida de Scrum y en tercer lugar la metodología basada en prototipos.
C3.5	**Según el presente TFC, la metodología mejor adaptada a los proyectos *web* es RAD, seguida de la Metodología basada en prototipos.**

6. Bibliografía

Ajani, S. *Extreme Project Management*. Writers Club Press. 2002.

Beck, K. *Extreme Programming Explained*. Addison-Wesley, Inc. 1999.

Boar, B. H. *Application Prototyping*. John Willey & Sons, Inc. 1984.

Charvat, J. *Project Management Methodologies*. John Wiley & Sons, Inc. 2003.

Cockburn, A. *Agile Software Development*. Highsmith Series Editors. 2000.

Fraga, M. *Metodología de Trabajo Ágil y Eficiente: El método Scrum aplicado a la gestión de proyectos en general*. Fundación Antonio de Nebrija. 2008.

Hurtado, T. *Ingeniería del Software*. Universidad Politécnica de Madrid. 1995.

McConnell, S. *Rapid Development*. Microsoft Press. 1996.

McConnell, S. *Software Estimation: Desmystifying the Black Art*. Microsoft Press. 2006.

Pressman, R. *Ingeniería del Software. Un enfoque práctico*. McGraw Hill. 2001.

Royce, W. *Software Project Management: A Unified Framework*. Addison-Wesley, Inc. 1998.

Schwaber, K. *Agile Project Management with Scrum*. Microsoft Press. 2004.

Shtub, A., Bard, J., Globeson, S. *Project Management. Process, Methodologies, and Economics*. Pearson Prentice Hall. 2005.

Stutzke, R. *Estimating Software-Intensive Systems*. Addison-Wesley. 2005.

Referencias *web*:

http://vapvarun.com/study/softE/teach%20yourself%20extreme%20programming%20in%2024%20hours/0-672-32441-5_2fch01lev1sec2.htm

Incluye diagramas de varios de los ciclos de vida descritos

http://www.clubdesarrolladores.com/articulos/mostrar/63-metodologia-scrum

Incluye el esquema de la metodología Scrum insertado

http://www.aimtec.cz/en/agile-software-development/

Incluye el esquema del paradigma Programación extrema

http://www.jknichols.co.uk/SL2.html

Esquema del Ciclo de vida en espiral

http://cflores334.blogspot.es/

Esquema de la Metodología basada en prototipos

http://www.compute-rs.com/es/consejos-339741.htm

Historia del Ciclo de vida en cascada

http://www.business-esolutions.com/islm.htm

Comparativa entre las diferentes metodologías de desarrollo de proyectos

http://www.CrystalMethodologies.org

Historia de las metodologías Crystal

http://www.asapm.org/asapmag/articles/A6_CrystalOrange.pdf

Esquema de actividades de la metodología Crystal orange / web

http://journal.info.unlp.edu.ar/journal/journal28/papers/JCST-Jun10-4.pdf

Proporciona un marco de trabajo para la evaluación de las metodologías ágiles

http://www.crystaltechnologies.com/carrier-solutions/life-cycle-support/project-planning

Puntos a tener en cuenta en la planificación de un proyecto implementado con Crystal orange / web

http://www.gantthead.com/process/processMain.cfm?ID=11347 (Es necesario ser miembro)

Esquema del seguimiento del proyecto que implementa la metodología RAD

http://www.c-mci.com/WhyCMCI.html

Esquema de la metodología RAD

Práctica Interacción Humana con Ordenadores primer semestre curso 2010/11 ITIG UOC

Punto de partida para los requisitos de alto nivel del proyecto ejemplo