Universidad de Oviedo

UNIVERSIDAD
DE LA RIOJA

UNIVERSIDAD DE OVIEDO

DEPARTAMENTO DE EXPLOTACIÓN Y PROSPECCIÓN DE MINAS
MASTER INTERUNIVERSITARIO EN "DIRECCIÓN DE PROYECTOS"

Trabajo Fin de Master

Modelo CMMI y métodos ágiles en la gestión de proyectos software

Autor: Juan Alonso Baldonedo

Director: Vicente Rodríguez Montequín

Junio 2017

Resumen

Este TFM busca abordar cómo dos áreas de la ingeniería de software aparentemente incompatibles como el modelo de mejora de procesos CMMI y las metodologías de desarrollo ágiles pueden ser utilizadas de manera cooperativa para mejorar el éxito de los proyectos software. Para poder comprender por qué puede ser interesante esta cooperación es necesario analizar cuáles son las características de los métodos ágiles, qué métodos existen y qué los diferencia de los tradicionales. Por otro lado, necesitamos entender también el proceso software y cómo ayudar CMMI a mejorarlo. Para ello debemos entender qué elementos componentes CMMI y cómo se estructuran. Solo de esta manera podremos llegar a entender cómo surge el CMMI ágil.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Introducción

Durante las últimas décadas, el software ha sido protagonista activo en numerosos cambios sociales, técnicos y culturales. Gracias a él, hemos experimentando una verdadera revolución en el mundo de las técnica, la medicina, las artes, las relaciones personales, y por supuesto ha sido el motor de la informática, sin duda una industria que ha redefinido buena parte de las actividades humanas.

Para lograr esta revolución, a la que aún estamos asistiendo, ha sido necesario que en torno al software surja una nueva forma de hacer ingeniería. La necesidad de desarrollar nuevas formas viene dado por la propia naturaleza intangible del software. En su libro "The Pragmatic Programmer" Andrew Hunt [1] destaca las diferencias entre la ingeniería tradicional y la ingeniería de software de la siguiente manera:

> *While you can plan a house or a skyscraper, such as an analogy for software engineering can hardly be used [...] Software is more like gardening. Plan the garden, plant the plants. Then see whats growing, remove weeds and plant new plants.*

A simple vista la visión de Hunt es la de que el software tiene más de proceso artístico que ingenieril. Sin embargo, si hacemos una lectura minuciosa de las palabras de Hunt, vemos que en realidad está describiendo una serie de procesos: *planificar*, *hacer*, *comprobar* y *actuar*. Estos no distan de ser similares a los procesos que nos encontraríamos en otras ramas de la ingeniería, pero en el caso del software veremos que los modelos tradicionales de trabajo presentan serios problemas e inconvenientes que deberán ser atajados con otros modelos que permitan afrontar las dificultades planteadas por la ya mencionada naturaleza intangible del software.

Cómo organizar los procesos de desarrollo software de forma que el resultado sea un software barato, de calidad, que satisfaga las expectativas del cliente, mantenible y obtenido en un tiempo razonable se ha convertido en el uno de los grandes retos de la ingeniería de software.

1.2. Motivación Trabajo Fin de Máster

Este trabajo fin de máster (TFM) se centra en el estudio de la mejora de procesos software mediante el modelo CMMI-DEV en organizaciones con metodologías de trabajo ágil. Para ello se ha estructurado el TFM en las siguientes secciones:

- **Capítulo 1: Introducción**. Breve descripción de los objetivos del trabajo, planteamiento inicial del trabajo y conceptos de partida empleados a lo largo del trabajo.
- **Capítulo 2: Ingeniería de Software**. Qué entendemos por ingeniería de software y cuáles son los procesos de desarrollo software existentes.
- **Capítulo 3: Métodos ágiles**. Introducción a los conceptos de desarrollo ágil y descripción de las principales metodologías.
- **Capítulo 4: Mejora de procesos y CMMI**. Visión general del proceso de mejora de procesos y funcionamiento detallado del modelo CMMI.
- **Capítulo 5: CMMI Ágil**. Cómo combinar las metodologías ágiles con el modelo CMMI.

1.3. Gestión de proyectos

PMI define proyecto como «*un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único*» [2]. De acuerdo con la Guía del PMBOK® la naturaleza temporal de los proyectos indica un principio y un final definidos. El final se alcanza cuando se logran los objetivos del proyecto o cuando se termina el proyecto porque sus objetivos no se cumplirán o no pueden ser cumplidos, o cuando ya no existe la necesidad que dio origen al proyecto. Temporal no necesariamente significa de corta duración. En general, esta cualidad no se aplica al producto, servicio o resultado creado por el proyecto; la mayor parte de los proyectos se emprenden para crear un resultado duradero.

La gestión de proyectos es la disciplina que se encarga de la inicialización, planificación, ejecución y control y cierre de trabajos que llevará a cabo de un equipo para conseguir completar un proyecto de acuerdo a un presupuesto, un plazo y una calidad esperada. La dificultad de la gestión de proyectos no reside en completar el proyecto sino en hacerlo cumpliendo estas tres restricciones.

La guía PMBOK de PMI [2] está basada en **procesos**, esto es, describe el trabajo a realizar como una serie de procesos. Entendemos por proceso el conjunto de acciones y pasos necesarios para alcanzar un objetivo. Para el PMBOK un proceso puede clasificarse en uno de los siguientes cinco grupos:

1. **Procesos de iniciación**. Encaminados a definir un nuevo proyecto o una nueva fase de un proyecto existente mediante la autorización para el inicio de ese proyecto o fase.

2. **Procesos de planificación**. Encaminados a establecer el alcance del proyecto, refinado de los objetivos y las acciones requeridas para lograr esos objetivos.

3. **Procesos de ejecución**. Encaminados a completar el trabajo definido en el plan del proyecto para satisfacer las especificaciones

4. **Procesos de monitorización y control**. Encaminados a seguir, revisar y regular el avance y rendimiento del proyecto, así como identificar cambios requeridos en el plan del proyecto, e iniciarlos.

5. **Procesos de cierre**. Encaminados a finalizar todas las actividades de todos los grupos de procesos para cerrar formalmente el proyecto o fase.

Esta aproximación es consistente con otros estándares de gestión como ISO9000 [3] o CMMI [4].

Existen otras guías y estándares para la gestión de proyectos, como los establecidos por IPMA con su ICB [5] o por el estándar ISO 21500:2012 [6]. Dado su enfoque a procesos y su amplio uso, emplearemos el enfoque PMI a lo largo del TFM.

1.4. Gestión de Proyectos Software

Las singularidades del software afectan directamente a la gestión de los proyectos software.

Esta complejidad en la gestión de proyectos no es un fenómeno novedoso. A finales de la década de los años 60 del siglo pasado, cuando la industria del software apenas contaba con unos pocos años de existencia, se acuñó el término "crisis del software" para describir la situación por la que pasaba el sector y que se manifestaba de diferentes formas [7]:

- Importantes sobrecostes
- Graves incumplimientos de plazos
- Software muy ineficiente
- Software de muy baja calidad
- Incumplimiento de requisitos
- Imposibilidad de mantener y gestionar el software existente
- Fracaso del proyecto y no entrega del software

El estándar Software Engineering Body of Knowledge (SWEBOK) [8], también conocido como ISO/IEC TR 19759:2005, recoge algunas razones que pueden ayudar a explicar el fracaso del desarrollo del software:

Año	Successful	Challenged	Failed
1994	16 %	53 %	31 %
1996	27 %	33 %	40 %
1998	26 %	46 %	28 %
2000	28 %	49 %	23 %
2002	34 %	51 %	15 %
2004	29 %	53 %	18 %
2006	35 %	46 %	19 %
2008	32 %	44 %	24 %
2010	37 %	42 %	21 %
2011	29 %	49 %	22 %
2012	27 %	49 %	22 %
2013	31 %	50 %	19 %
2014	28 %	55 %	17 %
2015	29 %	59 %	19 %

Cuadro 1.1: Resultados informe CHAOS

- Desconocimiento por parte del cliente de lo que es viable y de su dificultad.
- Dificultad por parte del cliente para comprender la complejidad de introducir cambios en el software existente.
- El aumento de la compresión y las condiciones cambiantes generan requisitos nuevos.
- Como resultado de los cambios, el software es desarrollado utilizando procesos iterativos en vez de una secuencia de tareas definidas y cerradas.
- La ingeniería de software tiene dos componentes opuestas: creatividad y disciplina. Mantener el equilibrio entre ambas no siempre es fácil.
- El grado de novedad al que se exponen los proyectos es normalmente muy alto.
- Alta tasa de cambio y evolución en la tecnología subyacente.

Debido a la alta percepción de fracaso en los proyectos software, la consultora especializada en gestión de proyectos software "Standish Group" realizó una investigación cuantitativa sobre el éxito de estos proyectos en el año 1994, el resultado fue el *Informe CHAOS* [9]. Este informe cosechó gran repercusión, no exenta de críticas por su opacidad y posibles sesgos [10] [11] [12], poniendo de manifiesto la necesidad de implantar nuevas formas de gestionar los proyectos software.

El informe ha venido publicándose con regularidad desde entonces. En él se hace una clasificación de los proyectos según su nivel de éxito en tres grupos:

- **Successful** o *exitoso*. El proyecto se completa en plazo, costes y con todas las funcionales definidas por el cliente.

- **Challenged** o *cuestionado*. El proyecto se completa, pero sin satisfacer por completo las expectativas de plazos, costes y/o funcionalidad.
- **Failed** o *fracasado*. El proyecto no llega a finalizarse y es cancelado.

Desde el primer informe, publicado en 1994, "The Standish Group" ha publicado con regularidad actualizaciones del mismo (ver tabla 1.1). Los resultados a lo largo de los años muestran una leve evolución favorable si atendemos al número de proyectos considerados exitosos. A pesar de esto, no deja de ser sorprendente el bajo número de proyectos considerados exitosos, apenas un tercio del total.

El informe no solo se centra en evaluar el número de proyectos exitosos, sino que también las metodologías empleadas, el tamaño de las organizaciones que los desempeña o la propia dimensión del proyecto software.

En cualquier caso, y dadas las dificultades para alcanzar el éxito en los proyectos software, las organizaciones deben proveerse de buenas técnicas, prácticas y elementos de gestión de proyectos software si quieren ser competitivas a la hora de producir software.

Capítulo 2

Ingeniería de Software

2.1. Introducción

Son muchas las actividades en las que la gente escribe programas para simplificar sus tareas, desde automatizar la contabilidad en hojas de cálculo, algoritmos empleados en actividades científicas, tareas repetitivas que se sustituyen por pequeños fragmentos de código, etc. Sin embargo, la mayor parte del software que se produce se hace de forma profesional, bien sea para ser utilizado dentro de un determinado producto (centralita de un coche, un ascensor, un electrodoméstico, etc.), para gestionar alguna actividad o servicio (servicios financieros, páginas web, buscadores de internet, etc.) o como producto mismo (programa CAD, suite ofimática, aplicaciones móviles etc.).

El software profesional es desarrollado por equipos de profesionales empleando técnicas, metodologías y prácticas que permiten especificar, diseñar y evolucionar dicho software de acuerdo a unas limitaciones de coste, tiempo y calidad establecidas. La **ingeniería de software** es la rama de la ingeniería que se encarga de este desarrollo profesional. Uno de los estudiosos más importantes de este campo, Barry Boehm, definía en 1979 [13] la ingeniería de software como:

> *the application of science and mathematics by which the capabilities of computer equipment are madeuseful to man via computer programs, procedures, and associated documentation.*

Esta definición contiene dos puntos interesantes, por un lado, extiende la tradicional noción de considerar software solo al código escrito, e incluye la documentación y procedimientos empleados. Por otro lado, introduce el concepto de que el software debe satisfacer de forma útil el propósito para el que es diseñado, es decir, la gestión del desarrollo software debe prestar suma importancia a que los requisitos y objetivos fijados se cumplen.

Sommerville [14] da una va incluso más allá y señala que la *ingeniería de software es una disciplina de la ingeniería que se ocupa de todos los aspectos relacionados con la producción de software, desde las etapas iniciales de especificación del sistema hasta el mantenimiento y desmantelamiento del sistema una vez haya*

concluido su ciclo de vida. De esta definición podemos destacar que:

1. *Disciplina de la ingeniería*. Los ingenieros crean cosas, para ello aplican teorías, métodos, y herramientas allá donde se necesiten. Sin embargo, lo hacen de forma selectiva y siempre tratando de encontrar solución a problemas donde incluso puede no haber teorías y métodos de actuación preestablecidos. Los ingenieros asumen que su trabajo se realiza dentro de organizaciones donde existen restricciones y limitaciones económicas y temporales que se deben tener muy en cuenta a la hora de realizar su trabajo.

2. *Todos los aspectos relacionados con la producción de software*. La ingeniería de software no solo debe ocuparse de los aspectos puramente técnicos del desarrollo software. Debe incluir otras actividades tales como la gestión del proyecto, la gestión de las herramientas de desarrollo, y de los métodos y teorías de producción de software.

La ingeniería busca satisfacer unas necesidades de acuerdo a un presupuesto, en un plazo determinado y con una calidad objetivo. Esto es lo que se conocer como el triángulo de gestión de proyectos, figura 2.1.

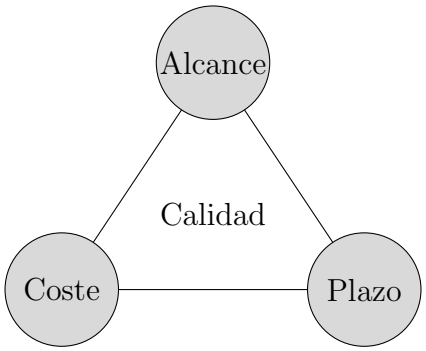

Figura 2.1: Triángulo de gestión de proyectos

En general los ingenieros de software buscarán adoptar un enfoque sistemático a su trabajo, ya que esta es la forma más efectiva de producir software de alta calidad. Gran parte del esfuerzo consistirá en elegir el método o métodos más apropiados en cada circunstancia, de tal forma que en una situación sea necesario seguir procedimientos más informales mientras que en otra la rigurosidad será imprescindible. Por ejemplo, el desarrollo de un portal web puede ser llevado a cabo de una manera más informal que el software de control de un avión.

El enfoque sistemático que se emplea en la ingeniería de software se suele denominar como **proceso software**. Un proceso software es la secuencia de actividades que conducen a la producción de un producto software. Podemos identificar cuatro actividades comunes a todo proceso software:

- **Especificación**. Donde el cliente e ingenieros definen el propósito del software que se va a producir, sus limitaciones y su operación.

- **Desarrollo**. Se lleva a cabo el diseño técnico y la programación.

- **Validación**. Se comprueba que el software cumple con los requisitos deseados por el cliente.

- **Evolución**. El software se modifica para atender a nuevas necesidades del cliente o del mercado.

Diferentes tipos de sistemas necesitarán diferentes tipos de desarrollo. Por ejemplo, un software de aviación con requisitos de tiempo real debe ser completamente especificado antes de que el desarrollo comience. Sin embargo, un portal de comercio electrónico suele especificarse paralelamente a su desarrollo. Consecuentemente, las cuatro tareas genéricas mencionadas se distribuirán de forma muy diferente dependiendo del proyecto.

2.2. El proceso software

Como ya hemos mencionado en la sección 2.1 existen diferentes tipos de procesos software, pero todos ellos de una u otra forma ejecutan cuatro actividades: "especificación", "desarrollo", "validación" y "evolución". En la práctica estas actividades son de enorme complejidad y para facilitar su compresión y aplicación se desomponen a su vez en sub-actividades, tales como "validación de requisitos", "diseño funcional", "pruebas unitarias", etc.

Cuando hablamos de procesos, normalmente describimos las actividades que los componen, tales como "diseñar el modelo de datos", "probar la interfaz de usuario", etc. y el orden en el que se suceden. Además de actividades la descripción de un proceso puede involucrar:

- **Productos**. Normalmente son el resultado de un proceso. Por ejemplo, el "proceso de diseño de arquitectura" tiene como resultado el producto "modelo de arquitectura software".

- **Roles**. Reflejan la responsabilidad del personal involucrado en el proceso. Ejemplos de roles pueden ser "jefe de proyecto", "analista", "programador" o "ingeniero de pruebas".

- **Pre- y pos-condiciones**. Son declaraciones que deben satisfacerse antes y después de ejecutar la actividad. Por ejemplo, antes de realizar el diseño funcional del software una precondición podría ser que los requisitos deban estar validados por el cliente y como pos-condición que el diseño sea plasmado en un modelo UML.

Los procesos software son complejos y como otras actividades intelectuales y creativas se basan en las decisiones y juicios de quienes toman parte. No existe un proceso único e ideal, no es raro encontrar organizaciones que hayan desarrollado sus propios procesos.

2.3. Actividades del proceso software

Los procesos software reales se constituyen como secuencias de actividades. Esta secuencia no tiene por qué ser lineal ni única, de manera que una actividad puede diferirse a lo largo del proyecto y realizarse en varios tramos. Cómo se distribuyen estas actividades dependen fundamentalmente del tipo de software y filosofía de gestión de proyecto.

A continuación, profundizaremos en las cuatro actividades básicas que componen el proceso software.

2.3.1. Especificación

La actividad de especificación, o también denominada **ingeniería de requisitos** es el proceso de compresión y definición de la funcionalidad y limitaciones del proyecto software. Es una actividad particularmente crítica ya que los errores cometidos en esta etapa son los de mayor impacto y coste.

El proceso de ingeniería de requisitos [15] persigue la definición de un documento que refleje los requisitos del sistema satisfaciendo las expectativas de todos los involucrados en el proyecto. Normalmente existe la necesidad de generar dos documentos, uno de alto nivel dirigido a los usuarios y otro más detallado para los desarrolladores.

Podemos identificar cuatro actividades principales en el proceso de ingeniería de requisitos:

1. **Estudio de viabilidad**. Estimación inicial para comprobar si las necesidades del cliente pueden ser cubiertas con la tecnología actual. Este estudio buscará saber si el sistema propuesto será viable desde un punto de vista empresarial y si puede ser desarrollado en plazo y coste. Este estudio debe ser rápido y barato para facilitar a la dirección la decisión de proseguir o interrumpir el proyecto.

2. **Educción y análisis de requisitos**. Es el proceso de recopilación de requisitos a través de observaciones del sistema actual, entrevistas con los usuarios finales, análisis, etc. Puede conllevar la elaboración de prototipos o pruebas de concepto que ayuden a comprender partes del sistema.

3. **Especificación de requisitos**. La información recopilada durante la fase de análisis es documentada en esta actividad. Pueden generarse por un lado *requisitos de usuario*, que son requisitos más abstractos y que describen lo que el cliente o usuario espera del sistema, y por otro los *requisitos de sistema* que son descripciones más detalladas del funcionamiento interno del sistema.

4. **Validación de requisitos**. Esta actividad consiste en la comprobación de los requisitos, de forma que cubran por completo las necesidades, sean realistas y consistentes.

Estas actividades no tienen por qué realizarse en estricta secuencia, y en la práctica nunca suelen estarlo. Además, es normal que conforme el proyecto avanza nuevos requisitos aparezca, u otros se vean modificados por lo que es una casuística a tener en cuenta.

2.3.2. Diseño e implementación

El diseño e implementación es el proceso en el que las especificaciones software son convertidas en código ejecutable. Involucra tareas de diseño técnico y programación, aunque si se utiliza un método incremental también conllevará afinar la especificación inicial.

El diseño consiste en establecer y describir la estructura del software, el modelado de datos y estructuras, los algoritmos y las interfaces con otros sistemas.

Buena parte del software se comunica con otro software mediante "interfaces" . Estas interfaces incluyen el sistema operativo, bases de datos, middleware, y otras aplicaciones. Todo ello compone lo que se denomina "plataforma software", es decir el entorno en el que se ejecuta el software. El entorno es también un elemento importante en el proyecto, ya que añadirá restricciones y fijará requisitos que deben ser tenidos en cuenta en el proceso de diseño. La *especificación de requisitos* es una descripción de la funcionalidad que el software debe proveer, sus requisitos y dependencias.

Las actividades en el proceso de diseño varían dependiendo del sistema bajo desarrollo. Un sistema industrial podrá tener un complejo diseño de interfaces, pero carecer de acceso a una base de datos. En la figura 2.2 se muestra un modelo abstracto del proceso de diseño software en el que se muestran cuatro actividades:

- **Diseño de arquitectura**. Identifica la estructura general del sistema, los principales componentes, sus relaciones y cómo se distribuyen.
- **Diseño de interfaz**. Se define el lenguaje común que utilizarán los diferentes elementos del sistema para comunicarse. Una vez definido los diferentes elementos podrán funcionar de manera desacoplada, es decir no necesitarán conocer su funcionamiento interno, tan solo la forma de comunicarse.
- **Diseño de componentes**. Se toma cada componente de la arquitectura y se diseña su funcionamiento.
- **Diseño de bases de datos**. Se diseñan las estructuras de datos y la forma en la que los datos son almacenados.

El resultado de estas actividades varía dependiendo del tipo proyecto. De nuevo si pensamos en proyectos de software crítico, estos necesitan de actividades de diseño de mayor rigurosidad y detalle. Si se utiliza la técnica de "diseño guiado por modelo" los resultados de estas actividades serán principalmente diagramas. Si por el contrario se emplea una metodología ágil de trabajo el resultado podría ser el propio código del software.

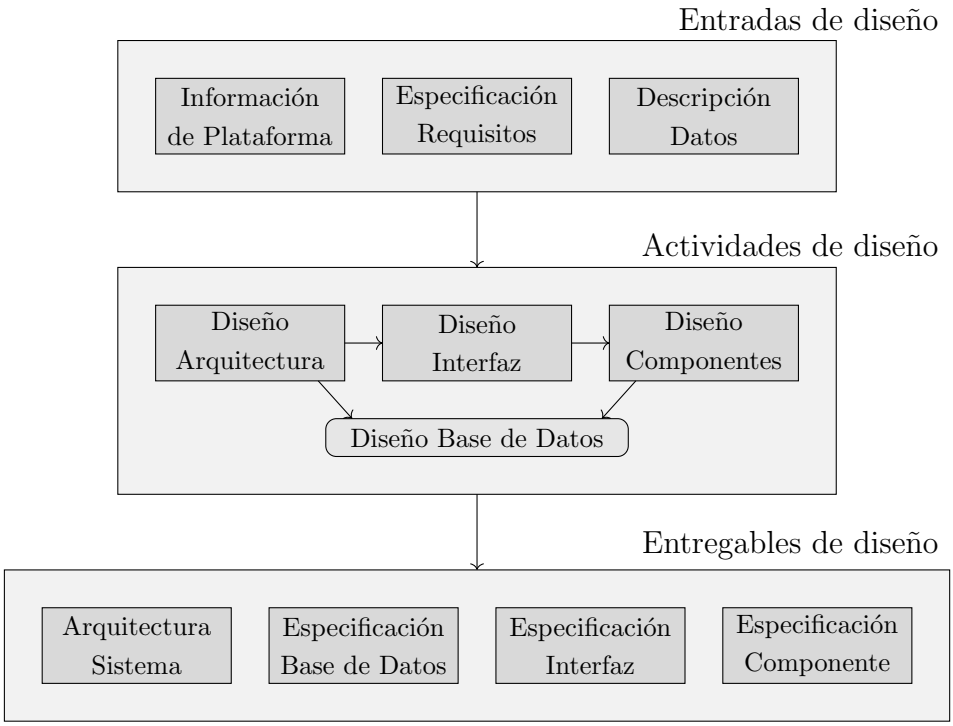

Figura 2.2: Modelo general del proceso de diseño

El desarrollo o programación del software se realiza tras el diseño del mismo. Típicamente se parte de un diseño inicial que se puede ir completando con detalles y casos no contemplados y que surgen durante la etapa de desarrollo, por lo que en general podemos hablar de un solapamiento de ambas tareas. Este solapamiento puede ser mayor o menor dependiendo del tipo de aplicación, un proyecto de software crítico en tiempo real tiende a definirse con mucho nivel de detalle en la etapa de diseño y por tanto el solapamiento es menor.

La programación por regla general no sigue ningún proceso, depende del programador o programadores. Qué partes se diseñan primero, qué enfoque o patrones utilizar son tareas generalmente más artesanales que formales.

Normalmente, el programador realiza pruebas sobre el código que produce con el fin de detectar defectos en el código.

2.3.3. Validación y verificación

La validación y verificación tiene como propósito asegurarse que el sistema cumpla con las especificaciones y expectativas del cliente. También se le denomina **control de calidad software**. Boehm [16] señala las que aunque validación y verificación son conceptos próximos y relacionados con la calidad software, dan respuesta a dos preguntas diferentes:

- **Verificación**. ¿Estamos haciendo el producto correcto?
- **Validación**. ¿Estamos haciendo el correctamente el producto?

La principal técnica de validación es lo que se conoce como *testing* o *pruebas software*, y es en la que se suelen incurrir más costes. Es común encontrarse con esquemas de tres niveles de testing, tal y como se muestra en la figura 2.3. Se comienza con pruebas de los componentes individuales, luego se prueban los componentes entre si y finalmente se realizan pruebas de sistema completo con información del cliente. Durante estas pruebas se encontrarán defectos en el software, y por tanto será necesario la toma de actuación para corregir estos defectos. De esta forma si se detecta un error en un componente por una programación defectuosa, el proceso de programación deberá ser puesto de nuevo en marcha y una vez concluido, el cambio probado otra vez hasta que la validación sea satisfactoria. Esto es un proceso iterativo.

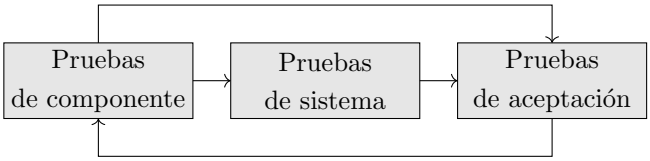

Figura 2.3: Fases de pruebas software

Distinguimos entonces tres etapas en el proceso de pruebas software:

1. **Pruebas de desarrollo**. También conocidas como *pruebas unitarias* o *unit testing*. Son pruebas llevadas a cabo por los propios programadores del componente software en ausencia de otros componentes del sistema. Si el componente requiere de interactuación con el mundo exterior, u otros componentes es habitual que estos sean simulados mediante *mocks*, de forma que el entorno de pruebas unitario sea totalmente controlado y pueda examinar el comportamiento de un determinado componente en una serie de situaciones representativas que permitan generalizar el comportamiento general del componente. Herramientas como JUnit y similares se han popularizado y extendido entre la comunidad de programadores para automatizar este proceso y facilitar la tarea de pruebas unitarias [17].

2. **Pruebas de integración**. Los diferentes componentes del sistema se prueban entre sí. Se buscan inconsistencias o fallos en las interfaces de comunicaciones o fallos motivados por errores en la coordinación de los componentes.

3. **Pruebas de aceptación**. Este tipo de pruebas son las últimas pruebas realizadas antes de que el sistema entre en servicio. Se realizan con el sistema completo y en las mismas condiciones y configuración que tendrá el sistema operativo. Si es posible el sistema se probará en situaciones reales. Este tipo de pruebas por lo general tienden a revelar fallos en las especificaciones. Las pruebas de aceptación son también conocidas como "Alpha Testing".

En el caso de que el producto final sea comercializado se suele añadir una fase más conocida como "Beta Testing". Esta fase consiste en la utilización del software por parte de un grupo de usuarios finales, quienes utilizarán la aplicación y reportarán los problemas vistos.

2.3.4. Evolución

La flexibilidad del software permite que la modificación del mismo una vez desplegado sea una actividad más del ciclo de vida. Esta flexibilidad viene de la mano del coste, modificar software es barato si se lo compara con el coste de la modificación hardware.

2.4. Cambios Software

En cualquier proyecto, independientemente de su naturaleza, la introducción de cambios y modificaciones es algo esperado y que se debe prever. Los proyectos software no solo no son ajenos a esta situación, sino que es incluso más habitual que en otro tipo de proyectos. Los cambios suelen venir motivados por cambios en el entorno del negocio. Además, la evolución de la técnica permite añadir nuevas prestaciones, arquitecturas o funcionalidades que en el pasado resultaban imposibles.

De esta forma, independientemente del modelo de desarrollo utilizado, es esencial que se articulen mecanismos que permitan modificar y evolucionar el software.

Los cambios implican aumentos en el coste del desarrollo software ya que supone rehacer partes del propio software. Este tipo de acciones se conoce como **retrabajos**. Si por ejemplo tras reevaluar los requisitos del sistema se encuentra alguna inconsistencia, uno o más requisitos deberán ser replanteados y rediseñados.

Existen dos enfoques relacionados que se usan para reducir los costes del retrabajo:

- Evitar el cambio, donde el proceso de software incluye actividades que anticipan cambios posibles antes de requerirse la labor significativa del retrabajo. Por ejemplo, puede desarrollarse un sistema prototipo para demostrar a los clientes algunas características claves del sistema. Ellos podrán experimentar con el prototipo y refinar sus requerimientos, antes de comprometerse con mayores costes de producción de software.

- Tolerancia al cambio, donde el proceso se diseña de modo que los cambios se ajusten con un coste relativamente bajo. Por lo general, esto comprende algunas formas de desarrollo incremental. Los cambios propuestos pueden implementarse en incrementos que aún no se desarrollan. Si no es posible, entonces tal vez un incremento tendría que alterarse para incorporar el cambio.

2.5. Paradigmas y modelos de proceso software

Hasta ahora hemos visto las diferentes actividades involucradas en el proceso software, a lo largo de esta descripción hemos recalcado que diferentes tipos de proyecto tienen diferentes maneras de abordar estas actividades. En esta sección

vamos a recorrer los principales modelos y paradigmas existentes bajo los que podemos encontrar la realización del proceso software. En el capítulo 3 veremos el desarrollo ágil.

2.5.1. El ciclo de vida de desarrollo software

Si existe un denominador común en las metodologías de desarrollo tradicionales que las distinga es la existencia de diferentes fases claramente delimitadas. De esta forma podemos hablar de **ciclo de vida de desarrollo de sistemas** o en inglés *Systems Development Life Cycle* (SDLC). Típicamente este ciclo viene dado por alguna de estas fases:

- **Análisis preliminar**. Supone realizar un estudio de viabilidad del sistema. Describe los costes y beneficios del proyecto. Se realiza un plan preliminar con recomendaciones y posibles alternativas.
- **Definición de requisitos**. Define el objetivo del proyecto, las funciones y operaciones que el sistema debe cumplir. Recopila los requisitos del usuario y/o cliente.
- **Diseño**. Describe la solución técnica capaz de satisfacer los requisitos propuestos en la etapa anterior.
- **Desarrollo**. Es la etapa donde se elabora el programa (programación).
- **Integración y pruebas**. Los diferentes subsistemas se prueban en conjunto y se buscan problemas, fallos o bugs que deban ser arreglas antes de permitir que el sistema entre en servicio.
- **Aceptación, instalación y despliegue**. La etapa final de desarrollo, donde el software es puesto en funcionamiento.
- **Mantenimiento**. A lo largo de la vida útil del software es necesario hacer ajustes, evolucionar ciertas partes o corregir fallos.
- **Evaluación**. También durante la vida útil del software puede ser necesario evaluar su rendimiento, prestaciones o necesidad para el negocio. El resultado de esta evaluación puede permitir a la dirección tomar decisiones como la mejora del software, su reemplazo o la substitución del mismo.
- **Desmantelado**. Es la fase que marca el fin de vida del software. Debe tener en cuenta cómo se procede al desmantelamiento del mismo sin causar perjuicio a la organización, departamento o personal dependiente del mismo.

No todas las metodologías y proyectos siguen rigurosamente estas fases, algunas pueden ser alteradas o modificadas. A continuación, veremos las metodologías tradicionales más populares.

2.5.2. Waterfall

El modelo *Waterfall* o **en cascada**, es uno de los modelos de desarrollo software más antiguos. Fue propuesto por Herbert D. Benington en el año 1956 durante el *Symposium on advanced programming methods for digital computers*, aunque formalizado unos años más tarde por Royce [18], Benington [19] o Bell [20]. En 1985 Departamento de Defensa de Estados Unidos (DoD) adoptó el modelo *Waterfall*, con ciertas modificaciones, para el desarrollo de software bajo el estándar *DOD-STD-2167A*.

Aunque existen varias interpretaciones de este modelo, si nos ajustamos al propuesto inicialmente por Royce [18] podemos distinguir cinco etapas, tal y como se muestra en la figura 2.4.

- **Recopilación de requisitos**.
- **Diseño**.
- **Implementación**.
- **Verificación**.
- **Validación**.

El modelo es similar al que se lleva a cabo en otros campos de la ingeniería, para pasar a una fase es necesario terminar con la anterior. Existen algunas variaciones de este modelo, algunas incluyen una validación al final de cada etapa que en caso de ser negativa implican el inicio de dicha fase.

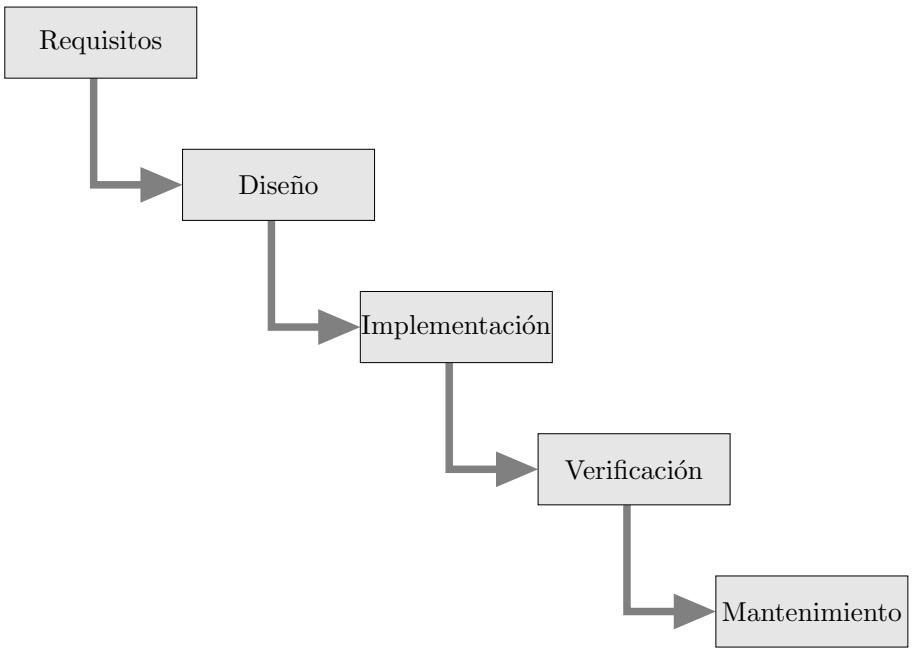

Figura 2.4: Modelo "waterfall" o de cascada

2.5.3. Modelo desarrollo en V

El modelo de desarrollo en V o **V-Model** se considera una extensión o evolución del modelo en cascada. En vez de moverse de arriba hacia abajo de forma lineal se dobla al llegar a la fase de programación volviendo a subir. El modelo en V busca relacionar cada fase del ciclo de desarrollo con una fase de pruebas tal y como se muestra en la figura 2.5

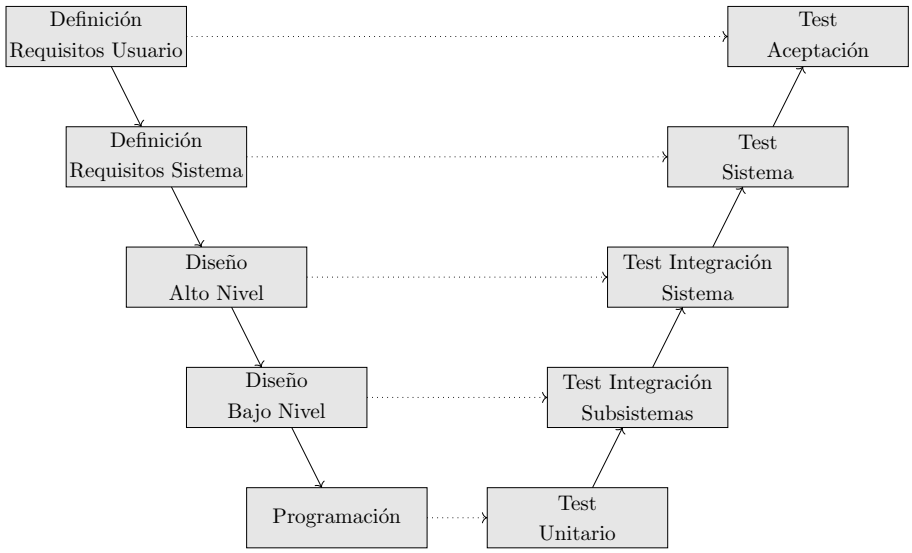

Figura 2.5: Modelo desarrollo en V

El lado izquierdo de la V representa la descomposición de las necesidades y la creación de las especificaciones del sistema. El lado derecho de la V representa la integración de las piezas y su verificación. V significa "Verificación y validación".

2.5.4. Prototipado

El prototipo es una versión inicial de un sistema software utilizada para demostrar conceptos, probar opciones de diseño y averiguar más sobre el problema y las posibles soluciones. El desarrollo de este prototipo se hace de forma incremental, lo más rápido posible de forma que los costes y plazos puedan ser controlados fácilmente y los involucrados puedan tomar decisiones rápidamente.

El prototipado software puede ser usado en el proceso de desarrollo software para ayudar a anticipar los cambios que puedan ser requeridos:

- Durante la fase de ingeniería de requisitos el prototipo puede servir de ayudar para la deducción y validación de requisitos del sistema.
- En la fase de diseño, un prototipo puede ser utilizado para explorar diferentes alternativas de diseño o servir de ayuda en la elaboración de la interfaz de usuario.

Los prototipos permiten a los usuarios extraer conclusiones, en ocasiones pueden ser utilizados para hacerse una idea de lo que se pretende conseguir o de las

limitaciones existentes. Pueden también ayudar al desarrollo de nuevos requisitos para el sistema.

Otra utilidad del prototipo es la comprobación de la viabilidad del software. Por ejemplo, se podría realizar un prototipo para comprobar la capacidad de procesamiento de mensajes de un middleware, o para evaluar el rendimiento de una base de datos.

El proceso de prototipado se expone en la figura 2.6, y comprende las fases:

1. **Objetivo**. El objetivo del prototipo debe ser establecido en primer lugar. Posibles objetivos podrían ser: comprobación de viabilidad, rendimiento, evaluar el coste y el plazo real del sistema completo, estudiar alternativas de diseño etc.

2. **Funcionalidad**. Una vez marcado el objetivo se debe establecer cuál es el alcance real del mismo, qué se debe incluir y qué se debe dejar fuera del prototipado.

3. **Desarrollo**. Fase donde se materializa el prototipo. El tipo de desarrollo del prototipo puede diferir en la metodología y forma al aplicado al sistema real. Además, es habitual que los prototipos no se ajusten a un guion de buenas prácticas de programación, por ejemplo, podría no implementarse un sistema de gestión de excepciones, eliminarse la documentación, o utilizar algoritmos más simples o rudimentarios si lo que se desea es realizar un prototipo de la interfaz de usuario que sirva al cliente para aprobar, descartar o rediseñar la interfaz final.

4. **Evaluación**. Es la fase en la que el prototipo se utiliza para cumplir con el objetivo inicial. Si el objetivo era realizar una interfaz de usuario en esta fase el cliente valida o corrige los elementos de la interfaz que deberán incorporarse en el software final.

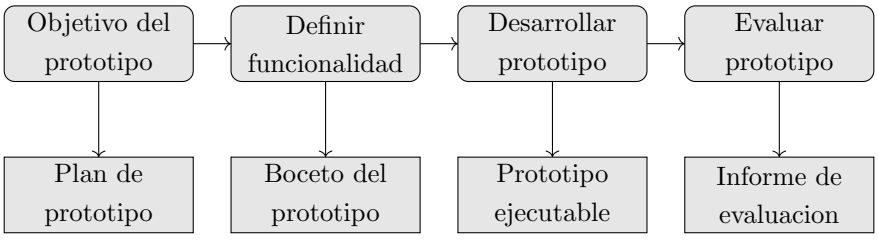

Figura 2.6: Proceso de desarrollo de prototipo

2.5.5. Método Incremental

El método incremental, ver figura 2.7, es un método de desarrollo software en el que el producto se diseña, ejecuta y prueba de forma incremental. Primero se hace una versión sencilla y reducida siguiendo una forma de trabajo en cascada. Cuando esta primera versión sencilla esta lista se vuelve sobre ella añadiendo

nuevas funcionalidades y repitiendo el proceso. La idea es aumentar la complejidad, partiendo de versiones sencillas a versiones más complejas hasta llegar al resultado deseado.

Es un método que goza de cierto uso dentro del desarrollo de software para gestión de sistemas [21].

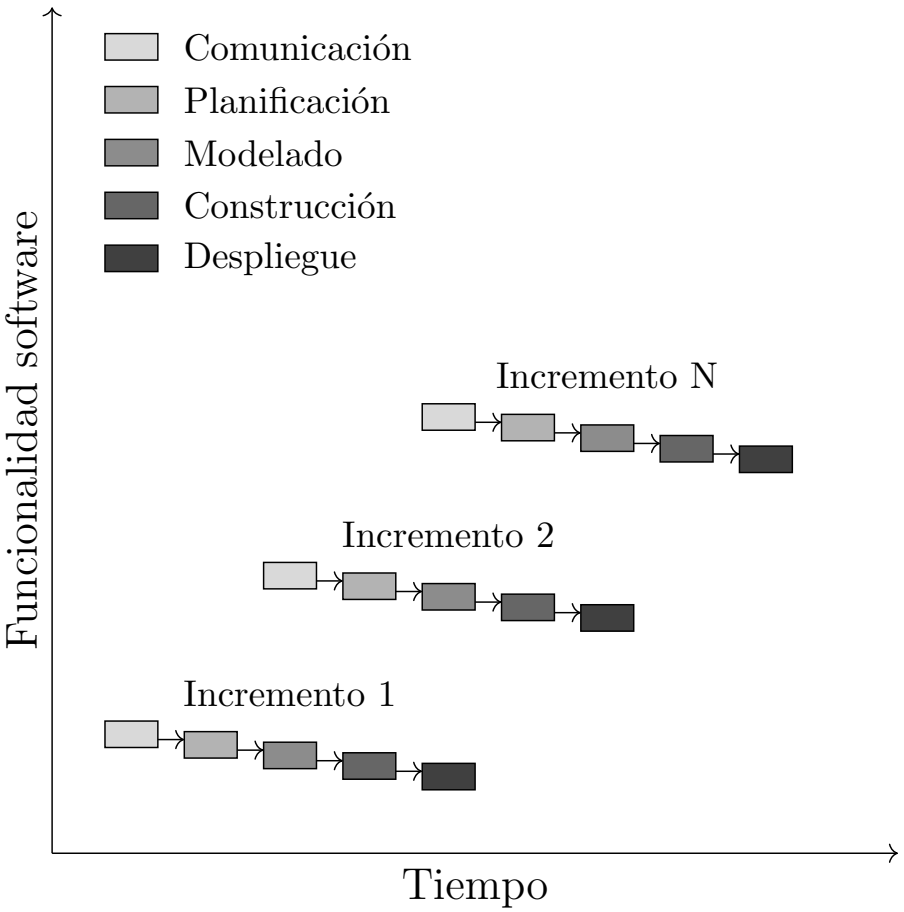

Figura 2.7: Modelo incremental

2.5.6. Método en Espiral

El método en espiral (figura 2.8), descrito por Barry Boehm en 1986 [22] [23], es un modelo en el que las actividades se conforman en una espiral, en la que cada bucle o iteración representa un conjunto de actividades. Las actividades no están fijadas a ninguna prioridad, sino que las siguientes se eligen en función del análisis de riesgo, comenzando por el bucle interior.

Cada iteración de la espiral está dividida en cuatro sectores:

1. **Establecimiento de objetivos**. Se definen objetivos específicos para la fase en cuestión. Se identifican los requisitos y restricciones existentes y se elabora un plan de acción.

2. **Análisis de riesgo**. Para cada uno de los posibles riesgos identificados se realiza un análisis detallado. Se toman los pasos necesarios para reducir estos

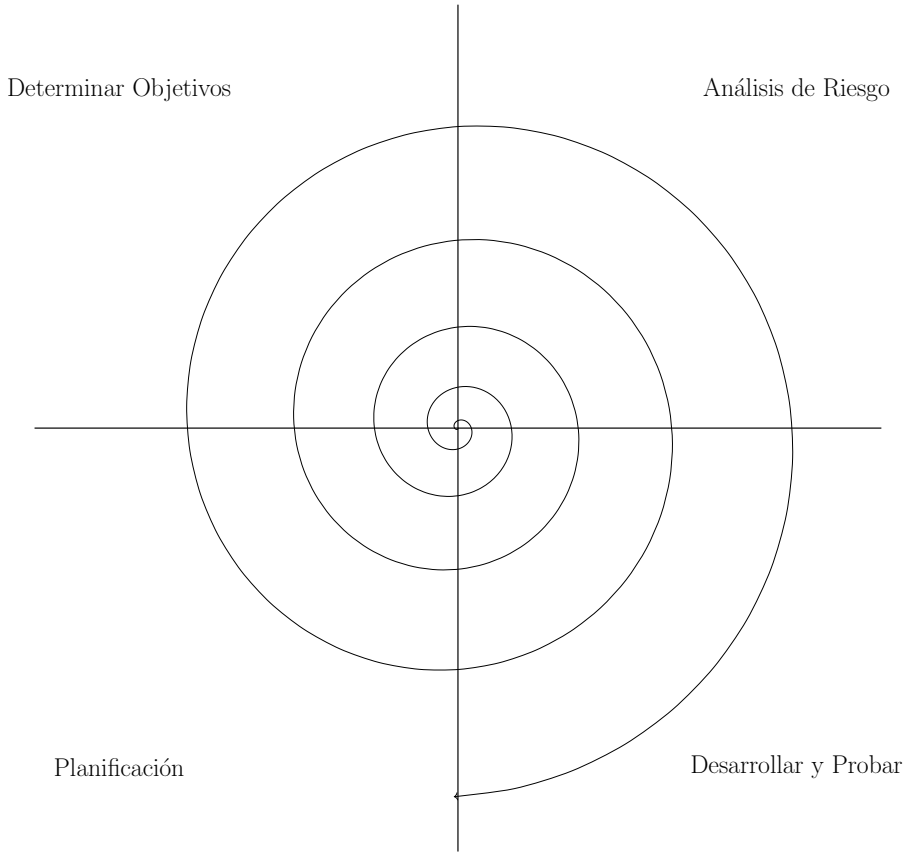

Figura 2.8: Modelo en espiral

riesgos. Por ejemplo, si existen dudas sobre la viabilidad de uno o varios requisitos se puede tomar la decisión de construir un prototipo.

3. **Desarrollo y validación**. Tras el análisis de riesgos se elige un modelo de desarrollo.

4. **Planificación**. Se revisa el proyecto y se toma la decisión de continuar o no con la siguiente iteración del proyecto.

La principal novedad de este proceso es la introducción del factor riesgo dentro del proceso. El riesgo es continuamente evaluado y mitigado.

2.5.7. Rational Unified Process (RUP)

El **Proceso Rational Unificado** o RUP (por sus siglas en inglés de *Rational Unified Process*) [24], es un proceso de desarrollo software moderno desarrollado a partir de los trabajos del modelo en espiral de Boehm [22] [23], los trabajos sobre UML [25] y el proceso asociado de desarrollo software unificado [26] [27].

RUP reconoce que los modelos de proceso convencionales presentan una sola visión del proceso. Por contra, RUP se describe desde tres perspectivas:

1. Una perspectiva dinámica que muestra las fases del modelo a través del tiempo.

2. Una perspectiva estática que presenta las actividades del proceso que se establecen.

3. Una perspectiva práctica que sugiere buenas prácticas a usar durante el proceso.

RUP es un modelo en fases que identifica cuatro fases discretas en el proceso software. Sin embargo, a diferencia del modelo en cascada, donde las fases se igualan con actividades del proceso, las fases en el RUP están más estrechamente vinculadas con la empresa que con las preocupaciones técnicas. Las fases son:

1. **Fase de inicio**. Esta fase tiene como propósito definir y acordar el alcance del proyecto con los patrocinadores, identificar los riesgos asociados al proyecto, proponer una visión muy general de la arquitectura de software y producir el plan de las fases y el de iteraciones posteriores.

2. **Fase de elaboración**. En la fase de elaboración se seleccionan los casos de uso que permiten definir la arquitectura base del sistema. Se realiza la especificación de los casos de uso seleccionados y el primer análisis del dominio del problema, se diseña la solución preliminar.

3. **Fase de desarrollo**. El propósito de esta fase es completar la funcionalidad del sistema, para ello se deben clarificar los requisitos pendientes, administrar los cambios de acuerdo a las evaluaciones realizados por los usuarios y se realizan las mejoras para el proyecto.

4. **Fase de transición**. El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados en las pruebas de aceptación, capacitar a los usuarios y dar el soporte técnico necesario. Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto.

El enfoque práctico del RUP describe las buenas prácticas de ingeniería de software que se recomiendan para el desarrollo de sistemas. Estos seis principios son:

1. **Desarrollo iterativo**.

2. **Administración de requisitos**.

3. **Uso de arquitectura basada en componentes**.

4. **Control de cambios**.

5. **Modelado visual del software**.

6. **Verificación de la calidad del software**.

El RUP no es un proceso adecuado para todos los tipos de desarrollo, por ejemplo, el software embebido.

2.5.8. Otros métodos

Existen otros métodos tradicionales, como "Clean Room" [28] desarrollado por IBM, "MIL-STD-498" del Departamento de Defensa de EEUU [29] o "DO-178C" creado por la industria aeroespacial [30]. En la práctica, suelen constituir variantes de los métodos anteriores especializados a sectores o industrias.

Capítulo 3

Métodos Ágiles

3.1. Introducción

El desarrollo ágil de software, basado en la construcción incremental e iterativa del mismo, ha cobrado gran popularidad en los últimos años. Sin embargo, esta forma de desarrollo tiene sus orígenes a principios de la década de 1960 [31], cuando dentro dela compañía IBM se comenzaron a hacer los primeros desarrolladores iterativos. A lo largo de las siguientes décadas, diferentes métodos de desarrollo más "ligeros" en cuanto a las formas empleadas fueron aplicados al desarrollo software [32] [33] [34].

A pesar de que como hemos visto los métodos de desarrollo incrementales e iterativos han estado presentes desde hace décadas, el concepto ágil, o lo que hoy se entiende por metodologías o desarrollo ágil es mucho más reciente. Este concepto abarca un conjunto heterogéneo de métodos, principios, recomendaciones y buenas prácticas que comparten una filosofía común que podríamos resumir en:

- Los procesos de especificación, diseño e implementación esta solapados. No existe una especificación detallada y la documentación de diseño es mínima o generada de forma automática por una aplicación documental. Los requisitos solo describen los elementos esenciales del sistema.

- El sistema se desarrolla en una serie de versiones. Los usuarios finales y los involucrados intervienen en la especificación y evaluación de cada versión. Pueden proponer nuevos cambios al software y especificaciones que deban integrarse en versiones sucesivas.

3.2. El manifiesto Ágil

Cuando hablamos de desarrollo ágil o *agile* lo solemos hacer para referirnos a la visión e ideas que toman como punto de partida el manifiesto ágil [35]. El manifiesto ágil nace en febrero de 2001 durante la reunión que un grupo de 17 expertos en ingeniería de software, entre los que se encontraban entre otros Jeff

Sutherland, Ken Schwaber, y Alistair Cockburn, mantuvieron en Utah para discutir sobre formas más ligeras de desarrollar software. El manifiesto dice lo siguiente:

> Estamos poniendo al descubierto mejores métodos para desarrollar software, haciéndolo y ayudando a otros a que lo hagan. Con este trabajo hemos llegado a valorar:
>
> - A los **individuos y su interacción**, por encima de los procesos y las herramientas.
> - El **software que funciona**, por encima de la documentación exhaustiva.
> - La **colaboración con el cliente**, por encima de la negociación contractual.
> - La **respuesta al cambio**, por encima del seguimiento de un plan.
>
> Aunque hay valor en los elementos de la derecha, valoramos más los de la izquierda.

El primer elemento que se destaca en el manifiesto es la importancia de los individuos que participan en el proyecto frente a los procesos en los que participan. Esta idea no implica que los procesos no tengan utilidad o no aporten valor al proyecto, sino que señala que son los individuos y no las herramientas o las formas de trabajar las que deben tomarse con mayor prioridad. Los procesos deben ser una ayuda y un soporte para guiar el trabajo. Deben adaptarse a la organización, a los equipos y a las personas; y no al revés. La defensa a ultranza de los procesos lleva a postular que con ellos se pueden conseguir resultados extraordinarios con personas mediocres, y lo cierto es que este principio es peligroso cuando los trabajos necesitan creatividad e innovación.

El software funcional es otro pilar básico. La idea se fundamenta en conseguir versiones funcionales en el plazo más corto posible, de esta forma se pueden introducir modificaciones y/o validar el desarrollo hasta un punto del mismo. Este punto en parte está ligado con el modelo de desarrollo de prototipos, visto en la sección 2.5.4. La documentación no debe eliminarse, pero si ser restringida a la mínima imprescindible.

Las prácticas ágiles están especialmente indicadas para productos difíciles de definir con detalle en el principio, o que si se definieran así tendrían al final menos valor que si se van enriqueciendo con retro-información continua durante el desarrollo. También para los casos en los que los requisitos van a ser muy inestables por la rapidez en la que cambia el entorno del negocio.

Para el desarrollo ágil el valor del resultado no es consecuencia de haber controlado una ejecución conforme a procesos, sino de haber sido implementado directamente sobre el producto. Un contrato no aporta valor al producto. Es una formalidad que establece líneas divisorias entre responsabilidades, que fija los referentes para posibles disputas contractuales entre cliente y proveedor.

En el desarrollo ágil el cliente es un miembro más del equipo, que se integra y colabora en el grupo de trabajo. Los modelos de contrato por obra no encajan.

Para un modelo de desarrollo que surge de entornos inestables, que tienen como factor inherente el cambio y la evolución rápida y continua, resulta mucho más valiosa la capacidad de respuesta que la de seguimiento y aseguramiento de planes pre-establecidos. Los principales valores de la gestión ágil son la anticipación y la adaptación; diferentes a los de la gestión de proyectos ortodoxa: planificación y control para evitar desviaciones sobre el plan.

Tras los cuatro valores descritos, los firmantes redactaron los siguientes, como los principios que de ellos se derivan:

- Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.
- Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
- Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los periodos breves.
- Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
- Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
- La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
- El software que funciona es la principal medida del progreso.
- Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica enaltece la agilidad.
- La simplicidad como arte de maximizar la cantidad de trabajo que no se hace, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto-organizan.
- En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.

Durante los últimos años, estas ideas han ido madurando y evolucionando. En 2005 algunos de los firmantes originales del manifiesto redactaron la *"declaración de independencia de la dirección de proyectos"* [36] con el objetivo de potenciar las habilidades de gestión de proyectos ágiles basándose en seis principios:

- Aumentar el retorno de la inversión centrándose permanentemente en la cadena de valor.
- Entregar resultados mediante compromisos fuertes con el cliente, compartiendo la propiedad del proyecto.
- Estar preparado para la incertidumbre y gestionarla a través de iteraciones, anticipaciones y adaptaciones.
- Dar rienda suelta a la creatividad y la innovación mediante el reconocimiento de los individuos que la hacen posible y creando el entorno que favorezca su aparición.
- Aumentar el rendimiento mediante la responsabilidad del grupo, compartiendo los éxitos y fracasos.
- Mejorar la efectividad y fiabilidad a través de estrategias específicas de acuerdo con las estrategias, prácticas y procesos.

En 2009 otro movimiento, dirigido por Robert C Martin escribió una extensión sobre los principios del desarrollo de software, el *"Software Craftsmanship Manifesto"* [37]. El fin de este manifiesto servir de guía al desarrollo de software. Es una respuesta de estos a los males percibidos en las prácticas establecidas de la industria, entre otros la priorización de las preocupaciones financieras sobre la responsabilidad del desarrollador.

En 2011 la *Agile Alliance* [38], organización creada por un grupo de los firmantes originales del manifiesto ágil con el fin de promocionar y divulgar las metodologías ágiles, crearon la *Guía de Prácticas Ágiles*. Esta guía es un compendio abierto de definiciones, términos, experiencias y guías ágiles aportados por la comunidad software.

3.3. Metodologías Ágiles

Dentro de lo que hemos visto como filosofía ágil podemos identificar una sería de metodologías de trabajo para el desarrollo de software. Algunas de estas metodologías de trabajo son anteriores al manifiesto ágil y, por tanto, ha sido tras su publicación cuando se las ha podido clasificar con esta etiqueta. En este capítulo veremos las principales metodologías ágiles.

3.3.1. Agile modeling

Esta metodología fue propuesta por Scott Ambler en 2000 y posteriormente publicada en su libro "Agile modeling: effective practices for extreme programming and the unified process" [39]. Su enfoque es el de conseguir una forma efectiva para modelar y documentar sistemas software. En esencia es un conjunto de valores, principios y prácticas para modelar software, que puede ser aplicado en el desarrollo de un proyecto software.

Agile Modeling (AM), no abarca todas las fases del proyecto y debe ser vista como un complemento a otras metodologías ágiles como la programación extrema que trataremos en la sección **??**, o metodologías tradicionales como RUP.

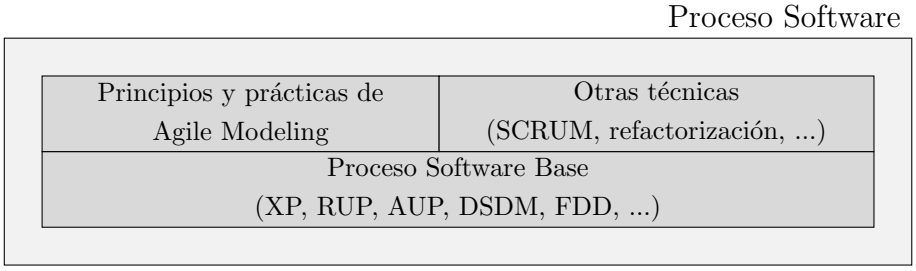

Figura 3.1: Proceso software con Agile Modeling

3.3.2. Adaptive software development

El **Adaptive software development** (ASD) o *Desarrollo Adaptativo del Software*, es una metodología ágil presentada por Jim Highsmith y Sam Bayer en 1998 [40]. ADS se centra en cómo construir software complejo mediante el trabajo colaborativo eficiente de los integrantes del proyecto que lo lleva a cabo de la organización que lo patrocina.

El Desarrollo adaptativo del software (ASD) proporciona un marco para el desarrollo iterativo de sistemas grandes y complejos. El método fomenta el desarrollo iterativo e incremental con el uso de prototipos.

ASD parte del siguiente axioma: *"los métodos en secuenciales en cascada o similares solo funcionan en entornos estables y bien conocidos"*. En la mayor parte de los proyectos los cambios en los requisitos y especificaciones serán frecuentes y por tanto el método de desarrollo deberá centrarse en cómo abordarlos. Tal y como se muestra en la figura 3.2, ASD funciona mediante un modelo de tres fases iterativo:

- **Fase de especulación**. Se inicia el desarrollo del proyecto, En ella se utiliza información como la misión del cliente, las restricciones del proyecto y los requisitos básicos para definir el conjunto de ciclos en el que se harán los incrementos del software.

- **Fase de colaboración**. Se busca que el equipo no solo se comunique o se encuentre completamente integrados, se desea que exista confianza, donde se puedan realizar críticas constructivas, ayudar si resentimientos y poseer un conjunto de actitudes que contribuyan al trabajo que se encuentran realizando.

- **El aprendizaje**. Permite mejorar el entendimiento real sobre la tecnología, los procesos utilizados y el proyecto. El aprendizaje individual permite al equipo tener mayor posibilidad de éxito.

Cada una de estas fases se unen entre sí para llevar a cabo diversas funciones, de manera rápida, y trabajando en equipo, para qué en un futuro, obtengamos un software eficiente.

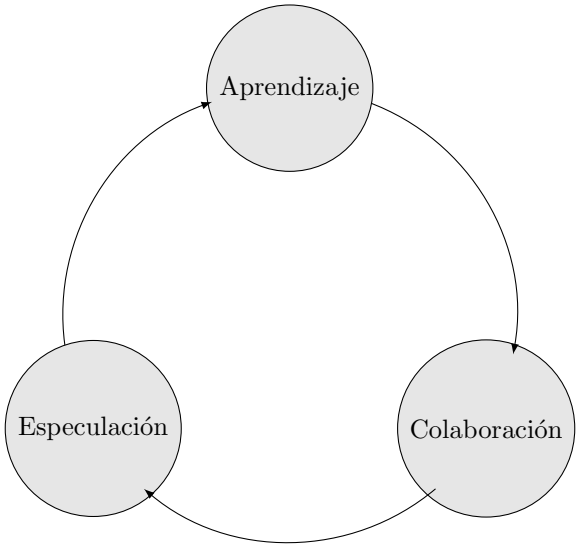

Figura 3.2: Proceso software con ASD

3.3.3. Feature-driven development

El *Feature-driven development* (FDD) o desarrollo basado en funcionalidades es un proceso de desarrollo incremental que aglutina a un conjunto de buenas prácticas en un todo. Estas prácticas están orientadas desde una perspectiva de funcionalidad de cara al cliente. El objetivo principal es entregar software funcional de forma periódica al cliente de forma iterativa hasta llegar al resultado deseado. Este método fue concebido por Jeff De Luca en 1999 [41].

FDD se fundamenta en cinco principios:

1. **Desarrollar un modelo global.** FDD comienza con un modelo de muy alto nivel que describa el alcance y entorno del proyecto. A continuación, se crean modelos detallados por cada área. Estos modelos los llevan a cabo pequeños equipos y se hace una revisión por pares de los mismos. Se eligen uno o una combinación de ellos como el modelo de cada área y se fusionan todos para construir el modelo global.

2. **Construir una lista de funcionalidades.** La información recopilada durante la fase anterior se emplea para identificar la lista de funcionalidades. Las funcionalidades son pequeñas descripciones que aportan valor al cliente y se expresan como “<acción>-<resultado>-<objeto>”.

3. **Planificar por funcionalidad.** Tras completar la lista de funcionalidades el siguiente paso es producir un plan de desarrollo, asignando responsabilidades de desarrollo a los equipos involucrados.

Hito	Porcentaje Asignado
Revisión del dominio	1 %
Diseño	40 %
Inspección del diseño	3 %
Programación	44 %
Inspección del código	10 %
Validación	1 %

Cuadro 3.1: Hitos y porcentaje asociado de una actividad FDD

4. **Diseñar por funcionalidad.** Para una funcionalidad o un pequeño conjunto de funcionalidades se diseñan los correspondientes diagramas.

5. **Construir por funcionalidad.** Tras el diseño funcional se pasa a la fase de desarrollo. Una vez desarrollado se prueba, y si la validación es satisfactoria la funcionalidad se integra en el desarrollo general.

Dado que las funcionalidades tienen un alcance reducido, completarlas es una tarea relativamente pequeña. Para llevar a cabo la supervisión del avance del proyecto es importante contabilizar el progreso de todas las funcionalidades. FDD define seis hitos por funcionalidad que deben ser completados secuencialmente. Los tres primeros hitos se completan durante la actividad de "diseñar por funcionalidad" y los últimos tres durante la fase de "construir por funcionalidad". Para ayudar al seguimiento, se le asigna un porcentaje de progreso a cada hito tal y como se muestra en la tabla 3.1

La filosofía FDD se asienta sobre un conjunto de buenas prácticas:

- **Modelado de dominio**. Es un modelo conceptual de todos los temas relacionados con un problema específico. En él se describen las distintas entidades, sus atributos, papeles y relaciones, además de las restricciones que rigen el dominio del problema.

- **Desarrollo por funcionalidad**. Cualquier funcionalidad que no pueda ser desarrollada en menos de dos semanas por culpa de su complejidad debe descomponerse en funcionalidades más simples.

- **Responsabilidad individual del desarrollo**. El programador es responsable de su código, esto es, de resolver los problemas que puedan surgir, de mantenerlo, de su rendimiento, etc.

- **Inspecciones**. Son tareas encaminadas a garantizar la calidad del y buen diseño del código. Fundamentalmente con el objetivo de encontrar *bugs* o cuellos de botella.

- **Gestión de configuración**. Es el conjunto de procesos destinados a asegurar la calidad de todo producto obtenido durante cualquiera de las etapas del desarrollo de un sistema de información (SI), a través del estricto control de los cambios realizados sobre los mismos y de la disponibilidad constante

de una versión estable de cada elemento para toda persona involucrada en el citado desarrollo.

- ***Builds* regulares**. La generación de ejecutables de forma regular ayuda a detectar problemas de forma más temprana y demostrar los avances realizados al cliente.

- **Visibilidad de resultados**. Reporte continuo del progreso en todos los niveles de gestión, de forma que los gestores puedan reorientar el proyecto si fuera necesario.

3.3.4. Extreme programming

La programación extrema o *extreme programming* (XP) es una metodología ágil desarrollada por Kent Beck [42]. Como otras metodologías ágiles propone entregas periódicas del software y desarrollo basado en ciclos cortos con el propósito de mejorar la eficiencia.

Otros rasgos de la programación extrema son: programación en parejas, revisión de código, pruebas unitarias de todo el código, evitar programar la funcionalidad no requerida y solo hacerlo cuando se necesite, una estructura plana de gestión, claridad y simplicidad en el código, esperar cambios en los requisitos del cliente según avance el tiempo y el problema sea comprendido mejor, y una comunicación frecuente entre los programadores y el cliente.

XP describe cuatro actividades que deben realizarse durante el ciclo de desarrollo software.

- **Programación**. XP argumenta que la actividad fundamental del ciclo de desarrollo software es la programación. Sin código no puede haber un producto funcional.

- **Pruebas**. El enfoque XP sostiene que, si un número reducido de pruebas es capaz de reducir problemas comunes en el software, un número elevado debe aumentar muy notablemente la calidad total del software. Por tanto, XP fomenta el uso de dos tipos de pruebas

  - **Pruebas Unitarias**. Son pruebas nivel de código.
  - **Pruebas aceptación**. Son pruebas a nivel funcional.

- **Escuchar**. Los programadores deben escuchar al cliente para satisfacer sus necesidades de negocio. Deben comprender estas necesidades lo suficiente para mostrar al cliente las alternativas técnicas, las posibilidades y las limitaciones.

- **Diseñar**. A pesar de que las actividades anteriores por si mismas serían suficientes para conseguir llevar a cabo el proyecto, en la realidad es necesario disponer de un diseño que permita dar coherencia al sistema. Un buen diseño evita redundancias, dependencias innecesarias y la escalabilidad posterior del software.

Los valores originales de la programación extrema son: simplicidad, comunicación, retroalimentación (feedback) y coraje. Un quinto valor, respeto, fue añadido en la segunda edición de Extreme Programming Explained [42]. Los cinco valores se detallan a continuación:

1. **Simplicidad**. La simplicidad es la base de la programación extrema. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento. Un diseño complejo del código junto a sucesivas modificaciones por parte de diferentes desarrolladores hace que la complejidad aumente exponencialmente. También se aplica la simplicidad en la documentación, de esta manera el código debe comentarse en su justa medida, intentando eso sí que el código esté autodocumentado. Para ello se deben elegir adecuadamente los nombres de las variables, métodos y clases. Los nombres largos no decrementan la eficiencia del código ni el tiempo de desarrollo gracias a las herramientas de autocompletado y refactorización que existen actualmente.

2. **Comunicación**. Desde el punto de vista del programador, el propio código es capaz de comunicar ideas y lo hará tanto mejor en la medida en la que este esté bien estructurado y sea claro. El código autodocumentado es más fiable que los comentarios ya que éstos últimos pronto quedan desfasados con el código a medida que es modificado. Debe comentarse sólo aquello que no va a variar, por ejemplo, el objetivo de una clase o la funcionalidad de un método. Las pruebas unitarias también son una buena fuente de comunicación, pues expresan aquello que se espera de un fragmento de código.

3. **Retroalimentación (*feedback*)**. La retroalimentación es una manera de llevar a cabo la monitorización y control del proceso software. La cooperación entre cliente y programador permite que el primero valide y conozca de primera mano la evolución del proyecto.

4. **Coraje o valentía**. Muchas de las prácticas implican valentía. Una de ellas es siempre diseñar y programar para hoy y no para mañana. Esto es un esfuerzo para evitar empantanarse en el diseño y requerir demasiado tiempo y trabajo para implementar el resto del proyecto. La valentía les permite a los desarrolladores que se sientan cómodos con reconstruir su código cuando sea necesario. Esto significa revisar el sistema existente y modificarlo si con ello los cambios futuros se implementaran más fácilmente.

5. **Respeto**. El respeto se manifiesta de varias formas. Los miembros del equipo se respetan los unos a otros, porque los programadores no pueden realizar cambios que hacen que las pruebas existentes fallen o que demore el trabajo de sus compañeros.

3.3.5. Agile Unified Process

Agile Unified Process (AUP) o Proceso Unificado Ágil, es una versión simplificada del Rational Unified Process (RUP) desarrollado por Scott Ambler [39]. Describe de una forma fácil de entender una aproximación al desarrollo de software empresarial mediante técnicas ágiles manteniéndose fiel a las técnicas y conceptos

definidos por RUP. Las técnicas ágiles empleadas por AUP incluyen *test-driven-development* o desarrollo dirigidos por pruebas, Agile Modeling (AM), gestión de cambios ágiles y refactorización de bases de datos para aumentar la productividad.

A diferencia de RUP, AUP cuenta con solo siete disciplinas:

- **Modelizado**. Consiste en comprender el negocio de la organización, el dominio del problema abordado por el proyecto e identificar una solución viable al problema planteado por el proyecto.
- **Implementación**. Transforma el modelo (o modelos) en código ejecutable a la vez que se desarrollan las pruebas básicas que verifiquen su funcionamiento.
- **Pruebas**. Esta fase consiste en la validación y verificación de del código implementando de acuerdo con los requisitos especificados.
- **Despliegue**. Planificación del despliegue y puesta en marcha del sistema.
- **Gestión de configuración**. Incluye gestionar el versionados de los componentes software y los cambios a lo largo del tiempo.
- **Gestión de proyecto**. Gestión propia del proyecto, incluyendo riesgos, recursos, plazos, etc.
- **Entorno**. Proporcionar a todos los involucrados las herramientas necesarias para llevar a cabo las tareas. Por ejemplo, recursos hardware.

Al igual que el resto de metodologías, tanto ágiles como tradicionales, AUP tiene su propia filosofía. Esta se puede resumir en:

- **El personal sabe lo que hace**. Los programadores no van a leer documentación detallada, pero si necesitan tener una visión de alto nivel y un entrenamiento ocasional.
- **Simplicidad**. Cualquier cosa puede ser descrita en pocas páginas y no en cientos o miles de páginas.
- **Agilidad**. AUP se adhiere a los principios ágiles descritos por la Agile Alliance.
- **Centrarse en actividades de valor**. El foco debe ponerse en aquello que es realmente relevante y no en aquello que sucede de forma extraordinaria.
- **Independencia de herramientas**. Las herramientas que se utilicen deben ser las más adecuadas para cada proyecto.
- **AUP a medida**. Las organizaciones o los proyectos deben definir en detalle cómo van a adoptar el modelo AUP de forma que se adapte mejor a sus necesidades.

Respecto a la forma en la que se generan versiones, AUP distingue entre dos tipos de iteraciones.

- **Iteración de desarrollo**. Esta versión es la entregada al equipo de pruebas o QA para realizar validaciones.
- **Iteración de producción**. Esta versión es la entregada al cliente para ser desplegada.

La figura 3.3 muestra un ejemplo de la evolución temporal del ciclo de *releases*. Este método constituye un pequeño avance respecto a RUP.

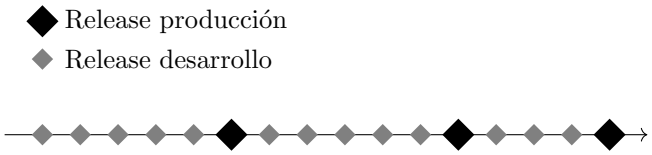

Figura 3.3: Esquema de *releases* en AUP

3.3.6. Dynamic systems development method

El *Dynamic systems development method* (DSDM) o método de desarrollo de sistemas dinámicos, es un método que provee un framework para el desarrollo ágil de software, apoyado por su continua implicación del usuario en un desarrollo iterativo y creciente que sea sensible a los requerimientos cambiantes, para desarrollar un sistema que reúna las necesidades de la empresa en tiempo y presupuesto.

Surgió a mediados de los años noventa para aportar cierta disciplina a el método *rapid application development* (RAD). En 2007 se convirtió en una aproximación genérica a la gestión de proyectos software. La última versión disponible, aparecida en 2014, reconoce la necesidad de operar junto con otros marcos de trabajo como ITIL, PRINCE2, y PMBOK.

DSDM consiste en tres fases, mostradas en la figura 3.4.

1. **Preproyecto**. Se identifican los proyectos candidatos, se consigue la financiación y el compromiso del proyecto. Puede haber un pequeño estudio de viabilidad previo.
2. **Ciclo de vida del proyecto**. Se componen a su vez de cinco etapas. Las dos primeras son secuenciales y complementarias. Después se realizan otras tres etapas de manera iterativa.
    a) **Estudio de viabilidad**. Se estudia la idoneidad de la metodología al proyecto.
    b) **Estudio de negocio**. Se involucra al cliente de forma temprana para comprender el problema a abordar.

c) **Iteración del modelo funcional**. Se produce una serie de prototipos incrementales que muestran al cliente la funcionalidad. Su propósito es el de recopilar requisitos adicionales y generar documentación de análisis.

d) **Diseño e iteración de la estructura**. Revisa la construcción de prototipos de la etapa anterior. Se diseña el sistema para su uso operacional.

e) **Implantación**. Tras la validación por parte de cliente el sistema se pone en servicio.

3. **Postproyecto**.

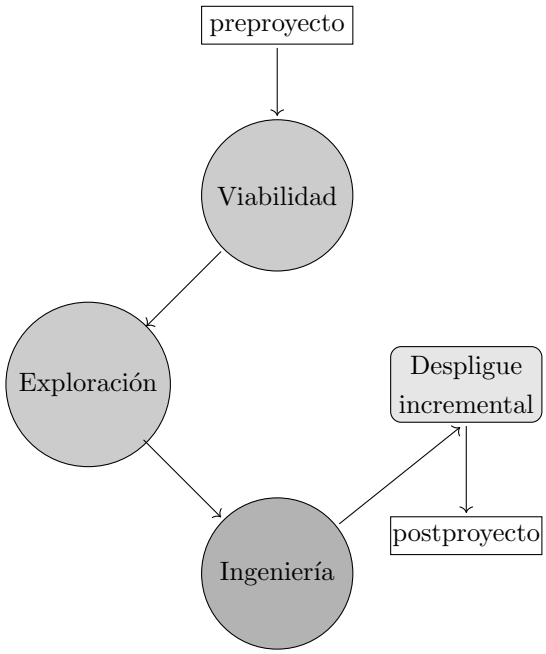

Figura 3.4: Fases de desarrollo DSDM

Hay 9 principios subyacentes al DSDM consistentes en cuatro fundamentos y cinco puntos de partida para la estructura del método. Estos principios forman los pilares del desarrollo mediante DSDM.

- Involucrar al cliente.
- El equipo del proyecto debe tener poder.
- El principal criterio de aceptación de entregables en DSDM reside en entregar un sistema que satisface las actuales necesidades de negocio.
- El desarrollo es iterativo e incremental, guiado por la realimentación de los usuarios para converger en una solución de negocio precisa.
- Todos los cambios durante el desarrollo son reversibles.
- El alcance de alto nivel y los requerimientos deberían estar alienados antes de que comience el proyecto.

- Las pruebas son realizadas durante todo el ciclo vital del proyecto.
- La comunicación y cooperación entre todas las partes interesadas en el proyecto es un prerrequisito importante para llevar un proyecto efectivo y eficiente.

Por último, DSDM establece una serie de roles cada uno con una responsabilidad definida: sponsor, ejecutivo, visionario, usuario, usuario embajador, usuario asesor, director de proyecto, coordinar técnico, líder ejecutivo, desarrollador, recopilador, facilitador, y especialista.

3.3.7. Lean software development

La metodología de desarrollo de software lean es una traducción de los principios y las prácticas de la forma de producir lean, hacia el área del desarrollo de software. Inicialmente, originado en el Sistema de Producción de Toyota y ahora, apoyado por una corriente que está surgiendo desde la comunidad Ágil. Este método ofrece todo un marco teórico sólido y basado en la experiencia, para las prácticas ágiles de gestión. El término *Lean Software Development* lo acuña el libro del mismo nombre escrito por Mary Poppendieck y Tom Poppendieck [43].

El desarrollo lean puede resumirse en siete principios, similares a los principios de *Lean Manufacturing*.

1. **Eliminar los desperdicios**. Todo aquello que no añade valor al software se considera desperdicio y debe eliminarse.
   - Código y funcionalidades innecesarias.
   - Retraso en el proceso de desarrollo software.
   - Requisitos innecesarios o poco claros.
   - Procesos burocráticos que no aportan valor.
   - Comunicación lenta.
2. **Amplificar el aprendizaje**. El desarrollo de software está siempre asociado con cierto grado de incertidumbre, los mejores resultados se alcanzan con un enfoque basado en opciones por lo que se pueden retrasar las decisiones tanto como sea posible hasta que éstas se basen en hechos y no en suposiciones y pronósticos inciertos. Cuanto más complejo es un proyecto, más capacidad para el cambio debe incluirse en éste, así que debe permitirse el retraso de los compromisos importantes y cruciales.
3. **Entregar tan rápido como sea posible**. Cuanto antes se entrega el producto final sin defectos considerables más pronto se pueden recibir comentarios y se incorporan en la siguiente iteración. Cuanto más cortas sean las iteraciones, mejor es el aprendizaje y la comunicación dentro del equipo.
4. **Capacitar al equipo**. Consiste en involucrar a los equipos en la toma decisiones. Los programadores deben ser capaces de transmitir hacia instancias

superiores y estas ser receptivas de manera que los problemas y soluciones se visibilicen y se puedan tomar las decisiones adecuadas.

5. **Construir integridad intrínseca**. El siempre exigente cliente debe tener una experiencia general del sistema. A esto se le llama percepción de integridad. Integridad Conceptual significa que los componentes separados del sistema funcionan bien juntos, como en un todo, logrando equilibrio entre la flexibilidad, sostenibilidad, eficiencia y capacidad de respuesta.

6. **Ver el todo**. Los sistemas de software hoy en día no son simplemente la suma de sus partes, sino también el producto de sus interacciones. Los defectos en el software tienden a acumularse durante el proceso de desarrollo por medio de la descomposición de las grandes tareas en pequeñas tareas y de la normalización de las diferentes etapas de desarrollo. Las causas reales de los defectos deben ser encontradas y eliminadas. Cuanto más grande sea el sistema, más serán las organizaciones que participan en su desarrollo y más partes son las desarrolladas por diferentes equipos y mayor es la importancia de tener bien definidas las relaciones entre los diferentes proveedores con el fin de producir una buena interacción entre los componentes del sistema.

3.3.8. Scrum

Scrum es una metodología ágil de desarrollo software muy popular hoy en día. El término fue introducido por Hirotaka Takeuchi y Ikujiro Nonaka en el contexto del desarrollo de productos en un artículo titulado *"The New Product Development Game"* en 1986 [44]. En este artículo los autores describían un nuevo enfoque para el desarrollo de productos comerciales que mejoraba la velocidad y flexibilidad a la hora de llegar al mercado. Se basaron en casos estudiados a partir de los métodos de trabajo usados en la industria japonesa del automóvil, de las máquinas fotocopiadoras y otras empresas manufactureras.

A comienzos de los años 90 las ideas niponas empezaron a aplicarse en la industria del software. En 1995 Sutherland y Schwaber presentaron el paper "Scrum methodology" [45] donde recogían los fundamentos de Scrum. Los siguientes años tras la publicación del paper original permitieron al Scrum inicial madurar y crecer. Tras la publicación en 2001 del manifiesto ágil [35] al año siguiente Schwaber junto con Mike Beedle escriben el libro *"Agile Software Development with Scrum"* [46] en el que se describe lo que hoy en día entendemos por Scrum. En 2002 Schwaber y otros crean la organización *Scrum Alliance* como herramienta para impulsar la metodología Scrum.

Scrum es un modelo de referencia que define un conjunto de prácticas y roles, y que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto.

Se definen tres tipos de roles dentro de Scrum:

- **Product owner**. Representa a todos los involucrados del proyecto, es la voz del cliente. Debe asegurarse que el equipo aporta valor al negocio. Es quien se encarga de escribir las historias de usuario, las prioriza en base a

su importancia y dependencias y las añade al *product backlog*. El *product owner* debe centrarse en el aspecto de negocio del proyecto y por tanto pasará la mayor parte del tiempo actuando de intermediario entre el equipo y el resto de involucrados. En consecuencia, se le debe exigir buenas dotes sintéticas y comunicativas y capacidad para guiar correctamente la dirección del desarrollo. Entre sus responsabilidades están:

  - presentar al resto de involucrados (y principalmente al cliente) no presentes en las reuniones el estado y avance del proyecto.
  - definir y anunciar las *releases*.
  - comunicar el estado del proyecto al resto de los integrantes del equipo de desarrollo.
  - organizar los hitos del proyecto.
  - negociar las prioridades, alcance, costes y plazos.
  - asegurarse que el *product backlog* está claramente definido y entendido por los miembros del equipo de desarrollo.

- **Equipo Scrum**. Es el responsable de entregar al final de cada *sprint* el desarrollo incremental previsto, también denominado *potentially shippable increment* (SPI). El equipo scrum debe constar de entre 3 y 9 personas para ser eficiente. Esto incluye a quienes se encarga de analizar, diseñar, desarrollar, probar, documentar, etc. El equipo Scrum se organiza independientemente, aunque esto no impide que exista algún tipo de interacción con la oficina de gestión de proyectos (PMO).

- **Scrum Master**. El Scrum es facilitado por un *Scrum Master*, cuyo trabajo primario es eliminar los obstáculos que impiden que el equipo alcance el objetivo del sprint. El *Scrum Master* no es el líder del equipo (porque ellos se auto-organizan), sino que actúa como una protección entre el equipo y cualquier influencia que le distraiga. El *Scrum Master* se asegura de que el proceso Scrum se utiliza como es debido. El *Scrum Master* es el que hace que las reglas se cumplan.

El proceso Scrum es iterativo (figura 3.5), cada iteración se la conoce como **sprint**. El sprint es un esfuerzo temporal finito y definido al comienzo (típicamente dos semanas). Cada Sprint comienza con una reunión de planificación del sprint en el que se define el *backlog*, que es un conjunto de requisitos de alto nivel priorizados que definen el trabajo a realizar. Durante esta reunión, el Product Owner identifica los elementos del Product Backlog que quiere ver completados y los hace del conocimiento del equipo. Entonces, el equipo habla con el Product Owner buscando la claridad y magnitud adecuadas para luego determinar la cantidad de ese trabajo que puede comprometerse a completar durante el siguiente Sprint. Durante el sprint, nadie puede cambiar el Sprint Backlog, lo que significa que los requisitos están congelados durante el Sprint.

Durante el desarrollo del Sprint se hacen reuniones diarias denominadas *Daily Scrum* o *Stand-up meeting* donde de manera muy breve, esta reunión debe durar unos pocos minutos, todos los miembros del equipo se ponen al día del estado del

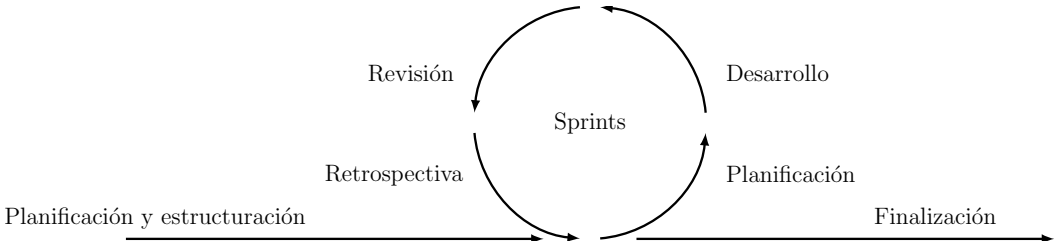

Figura 3.5: Ciclo de vida Scrum

desarrollo del proyecto. Durante esta reunión los miembros del equipo de desarrollo toman la palabra para responder brevemente a las siguientes preguntas:

- ¿Qué has hecho desde ayer?
- ¿Qué es lo que haré hoy?
- ¿Has tenido algún problema que te haya impedido alcanzar tu objetivo?

El objetivo de estas reuniones es que cada miembro del equipo sepa si se están cumpliendo los plazos marcados en el Sprint actual.

Al finalizar un Sprint se realizan dos reuniones una de *revisión del sprint* o *Sprint Review* en la que se abordan los siguientes puntos:

- Revisión del trabajo completado y no completado.
- Cómo presentar el trabajo completado a los interesados, por ejemplo, una demostración.
- Analizar por qué no se pudo completar parte del trabajo.

Esta reunión es más larga que las diarias y se recomienda que su duración sea de una hora por semana de trabajo. La otra reunión realizada es la reunión de *Retrospectiva del Sprint* o *Sprint Retrospective* en la que todos los miembros del equipo dejan sus impresiones sobre el sprint recién superado. El propósito de la retrospectiva es realizar una mejora continua del proceso.

Scrum es uno de los métodos ágiles más adoptados y que goza de mayor popularidad. Esta popularidad se debe en parte a los beneficios que se le atribuyen:

- **Flexibilidad**. Scrum se adapta muy bien a cambios en los requisitos. El marco de trabajo está diseñado para adecuarse a las nuevas exigencias que implican proyectos complejos.
- **Reducción del Time to Market**. Esta característica le hace ser muy atractivo en el mundo *startup* donde el tiempo de desarrollo es "tiempo perdido" y lo que prima es disponer de un producto viable para ser comercializado.
- **Buena calidad software**. El trabajo metódico y la necesidad de obtener una versión de trabajo funcional después de cada iteración, ayuda a la obtención de un software de alta calidad.

- **Productividad**. Las reuniones cortas, la auto organización del equipo y la falta de burocracia consigue que el equipo de desarrollo emplee la mayor parte del tiempo en desarrollar.

- **Maximización de la inversión**. Se tiende a producir aquello que se necesita y por tanto se evita crear componentes inútiles para el negocio.

- **Predicciones de tiempos**. A través de este marco de trabajo se conoce la velocidad media del equipo por sprint, con lo que es posible estimar de manera fácil cuando se podrá hacer uso de una determinada funcionalidad que todavía está en el Backlog.

- **Reducción de riesgos**. El hecho de desarrollar, en primer lugar, las funcionalidades de mayor valor y de saber la velocidad a la que el equipo avanza en el proyecto, permite despejar riesgos efectivamente de manera anticipada.

3.3.9. Kanban

Al igual que Scrum, Kanban surge en Japón en el ámbito de la mejora de procesos de producción en el sector del automóvil. En particular se inspira en el método *Toyota Production System* y en el *Lean manufacturing*. Este sistema persigue gestionar como se van completando las tareas de manera visual.

En su libro *Kanban* [47], David Anderson describe la evolución del mismo desde su inicio a partir de un proyecto de 2004 por parte de Microsoft [48], pasando por la incorporación más tarde del *kanban pull system* hasta un proyecto llevado a cabo por parte de la empresa Corbis en el cual se habla explícitamente del método Kanban.

El método Kanban divide el trabajo en partes, y cada parte se escribe en tarjetas que se colocan sobre un tablero, figura 3.6. Las tarjetas incorporan información variable tal como requisitos, estimación temporal, recursos, etc. El objetivo es que se visualice el trabajo a realizar, las prioridades, tiempos y estados de la tarea y en qué está trabajando cada individuo. El tablero tiene tantas columnas como estados puede tener una parte o tarea.

A diferencia de Scrum en Kanban no se asignan roles para dotar al equipo de mayor dinamismo. Tampoco se fijan reuniones diarias.

Este método ha sido criticado por algunos autores que no lo consideran un método ágil en sí mismo sino una herramienta que puede complementar a otras metodologías como la programación extrema o Scrum. De la combinación de Scrum y Kanban ha aparecido recientemente una nueva metodología denominada *Scrumban* que analizaremos en la siguiente sección. Tal vez sea este el motivo por el cual no es solo empleado en el desarrollo software y se ha comenzado a emplear en otros sectores como el marketing, recursos humanos, contabilidad etc.

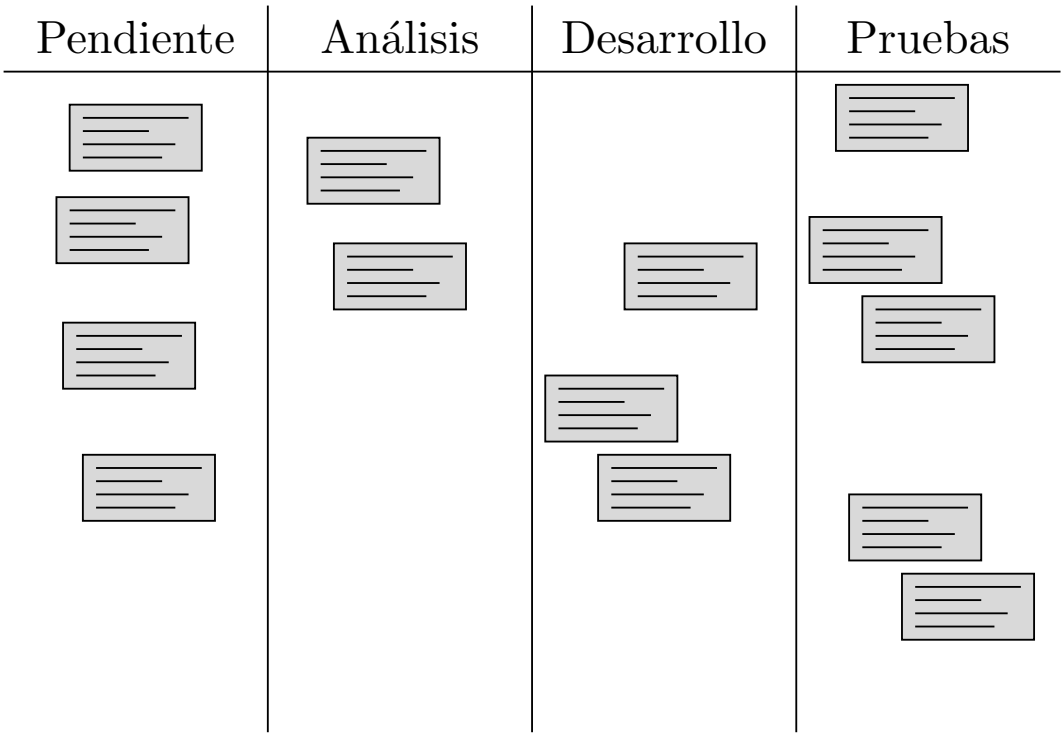

Figura 3.6: Tablero de trabajo Kanban

3.3.10. Scrumban

Como su propio nombre da a entender *Scrumban* es una metodología de desarrollo ágil fruto de la combinación de las metodologías *Scrum* y *Kanban*. El primer artículo que hace referencia explícitamente a *Scrumban* data del año 2008 [49].

Esta metodología es fundamental un marco de gestión que surge cuando el equipo de desarrollo emplea Scrum como método de trabajo y Kanban para mostrar y visualizar el trabajo pendiente, en desarrollo, en pruebas o finalizado. Es una forma de visualizar en todo momento la situación del Sprint.

Scrumban modifica ligeramente la filosofía tradicional de Scrum:

- Reconoce la importancia del rol de gestión.
- Fomenta la creación de equipos especializados.
- Se priorizan las tareas siguiendo criterios económicos.
- Aplica las leyes de flujo y teoría de colas a las tareas.

También difiere de la filosofía Kanban en cuanto a que:

- Se organiza en equipos
- Reconoce y valora la fragmentación del proyecto en intervalos temporales bien definidos.
- Reconoce la importante de las reuniones como forma de mejora continua.

Scrumban resulta ser muy adecuado para proyectos de mantenimiento software, proyectos en las que las históricas de usuarios varíen frecuentemente o en los que la complejidad y el riesgo sea alto.

Capítulo 4

CMMI

4.1. Introducción

A mediados de los años 80 del siglo pasado el Departamento de Defensa (DoD) de Estados Unidos decide crear el Instituto de Ingeniería de Software o *Software Engineering Institute* (SEI). El SEI se constituye como un centro de investigación dentro de la Universidad Carnegie Mellon, en Pittsburgh. Entre las áreas de competencia del SEI, el Departamento de Defensa marca como prioritaria la búsqueda de un modelo que permita evaluar las capacidades de los contratistas software del ejército americano.

Los trabajos llevados a cabo en el SEI dieron lugar al *Software Capability Maturity Model* o *Modelo de Capacidad y Madurez* (CMM) [50] [51]. Este modelo ha sido una aportación fundamental a la ingeniería de software en general y a la forma en la que se realiza la mejora de los procesos software de una organización.

CMM fue tan solo el comienzo, y tras él, han surgido otros modelos de madurez, como el *People Capability Maturity Model* (PCMM) [52] que se centra en la capacidad y continua mejora de la gestión y desarrollo de los activos humanos, o el *Systems Engineering Capability Model* [53] que se centra en capacidad de los sistemas.

Otras organizaciones igualmente han desarrollado marcos de trabajo para la mejora de procesos similares. Un ejemplo es el enfoque SPICE [54], un modelo de capacitación más flexible que el modelo CMM del SEI.

Otro ejemplo de modelo de madurez fue el "Bootstrap Project" [55], impulsado desde las instituciones europeas y que fue usando por la Agencia Espacial Europea para evaluar la calidad de sus procesos a lo largo y ancho de la organización.

A finales de los 90, el amplio abanico de modelos de madurez y capacitación existentes hizo que desde el SEI se plantease la necesidad de disponer de un nuevo modelo que recogiese la experiencia acumulada por todos ellos y crear así un nuevo estándar. De esta forma nace el marco de trabajo CMMI [56] [57] [58] [59] [4] como sucesor del CMM.

El modelo CMMI surge con la idea de ser aplicado a la mejora de procesos en un amplio espectro de organizaciones. El modelo es lo suficientemente complejo,

consta de más de 1000 páginas, para que se pueda ajustar a las necesidades de organizaciones de muy diversa naturaleza.

4.2. Mejora basada en procesos

La mejora basada en procesos o *Model-Based Process Improvement*, consiste en la utilización de un modelo que guíe la mejora de los procesos de una organización.

Entederemos por **proceso software** la secuencia de actividades que, cuando se ejecutan, producen como resultado un sistema software.

La mejora de procesos nace a partir de los trabajos de la gestión de calidad de autores como Demming [60], Crosby [61] y Juran [62] y está dirigido al aumento de la capacidad de trabajo de los procesos. La **capacidad de un proceso** es la habilidad de un proceso para producir un resultado dentro de los límites fijados en las especificaciones. Si la capacidad de un proceso aumenta lo suficiente, este se convierte en predecible y por tanto medible, y las causas más significantes de mala calidad pueden ser eliminadas o controladas. El aumento de las capaciones de los procesos empleados dentro de una organización hace que la **madurez** de la misma aumente. Conseguir altos niveles de madurez implica un fuerte compromiso por parte de la dirección de la organización y una visión estratégica a largo plazo, además de cambios en la manera de actuar y proceder en todos los niveles organizativos.

En los trabajos intelectuales, y el software es uno de ellos, la calidad resultante depende de su diseño. Existen cuatro factores importantes que afectan a la calidad:

- calidad de procesos
- calidad del personal
- coste y plazos
- tecnología de desarrollo

La influencia de cada uno de estos cuatro factores depende del tamaño del proyecto. Para sistemas muy grandes que incluyen a su vez otros subsistemas más pequeños, desarrollados en ocasiones por equipos externos en diferentes localizaciones, el principal factor que afecta a la calidad es el proceso software. Los principales factores en grandes proyectos software son la integración, la gestión del proyecto y la comunicación. Normalmente existe una mezcla de habilidades y experiencia entre los miembros del proyecto, y dado que el proceso de desarrollo se lleva a cabo durante años, existe una alta rotación entre los miembros del equipo de desarrollo. Incluso el equipo podría cambiar por completo entre el inicio y el final.

En proyectos más pequeños, donde solo existe un número reducido de desarrolladores, la calidad del equipo es fundamental. Además, en este tipo de proyectos, el tiempo de desarrollo (programación) consume la mayor parte del tiempo de

proyecto, por lo que una buena tecnología de desarrollo afecta notablemente a la calidad resultante.

Independientemente del tamaño si un proyecto está mal dimensionado en coste y/o plazos la calidad resultante se verá afectada de alguna forma.

4.2.1. El proceso de mejora de procesos

Los procesos software son observables en cualquier organización que produzca software, independientemente de su tamaño. Estos procesos son de muy diferentes tipos dependiendo del grado de formalidad del proceso, del tipo de producto desarrollado, del tamaño de la organización, etc. No existe un estándar o proceso software ideal. Cada individuo, compañía u organización puede desarrollar el suyo propio acorde a sus necesidades, conocimientos, tamaño, experiencia, clientes, tipo de software desarrollado, mercado al que se dirige o cultura empresarial.

La mejora de procesos, por tanto, no implica adoptar una determinada forma de trabajo, uso de una herramienta, o el seguimiento de unas reglas dadas. Aunque existan organizaciones con necesidades software similares, pensemos por ejemplo, en administraciones públicas desarrollando una plataforma para el pago de tributos, todas ellas tendrán particularidades locales que las afectan, por ejemplo diferentes regulaciones a las que se ven sometidas. Por tanto, rara vez una mejora de procesos en una determinada organización es exportable a otra. Se deben considerar los factores locales a la organización a la hora de mejorar sus procesos.

Por otra parte, se debe considerar los atributos de los procesos que se deseen mejorar. Si el objetivo es mejorar la estimación de plazos de los proyectos software se podrían por ejemplo introducir nuevas actividades o herramientas que modifiquen los pasos a seguir para una estimación más exacta. En la tabla **??** se presentan algunos ejemplos de atributos de procesos que pueden ser susceptibles de mejora. Estos atributos suelen estar relacionados, así, si un proceso tiene gran "visibilidad" es probable que sea de fácil "comprensión": un observador externo podría inferir las actividades llevadas a cabo con solamente ver las salidas del proceso. En cambio, un proceso con poca "visibilidad" podría estar inversamente relacionado con la "rapidez". Hacer el proceso visible implica que los involucrados generen información sobre el mismo, lo que afecta directamente al tiempo que se necesita para finalizarlo.

El proceso de mejora de procesos es cíclico e implica tres subprocesos tal y como se muestra en la figura 4.1:

- **Medición del proceso**. El objetivo es medir atributos de forma que se puedan establecer objetivos de mejora cuantitativos.

- **Análisis del proceso**. El proceso se audita, identificando sus puntos débiles y los posibles cuellos de botella. En ese punto se pueden desarrollar los modelos del proceso (también conocidos como mapas del proceso). El análisis puede estar enfocado a una serie de atributos como la "robustez" o la "comprensión" del proceso.

Característica del proceso	Aspectos clave
Comprensión	¿Hasta que punto el procesa esta definido y cómo de fácil es comprender la definición del proceso?
Estandarización	¿Hasta que punto el proceso esta basado en un proceso genérico? ¿Esta siendo utilizado en otros partes de la organización?
Visibilidad	¿Las actividades del proceso culminan en un resultado clave de forma que este sea visible?
Medible	¿Existen atributos en el proceso que le hagan susceptible de ser medido?
Robustez	En caso de adversidades de algún tipo ¿se puede continuar con el proceso? ¿en qué situaciones no se puede?
Mantenimiento	¿Puedo el proceso evolucionar y reflejar cambios requeridos por la organización?
Rapidez	¿Cuánto tiempo es necesario para completar el proceso desde qué se especifica su misión?

Cuadro 4.1: Atributos genéricos de procesos

- **Cambios en el proceso**. Por último, se deben de elaborar propuestas de cambios que mitiguen las debilidades y cuellos de botella detectados.

La mejora de procesos es una tarea de largo plazo que puede llevar semanas, meses o incluso años dependiendo del tamaño del proyecto y la organización. Es también una tarea continua, sin fin, y que evoluciona de acuerdo con los cambios y necesidades que aparecen en las organizaciones.

4.2.2. Medición de Procesos

La medición de procesos permite analizar de manera cuantitativa un proceso, por ejemplo, midiendo el tiempo necesario para generar y realizar las pruebas de validación software.

Se pueden establecer tres tipos de métricas de procesos:

1. **Tiempo para completar un proceso**. Este puede ser el tiempo total necesario para completar un proceso. Pueden existir variantes del mismo como el tiempo necesario para completar el proceso en función de los recursos empleados, el tiempo que emplean un determinado grupo de ingenieros en el proceso, etc.

2. **Recursos empleados**. Los recursos pueden ser expresados en número de personas por día necesarios, la capacidad de computación utilizada, etc.

3. **Número de ocurrencias de eventos**. Representa el número de veces que

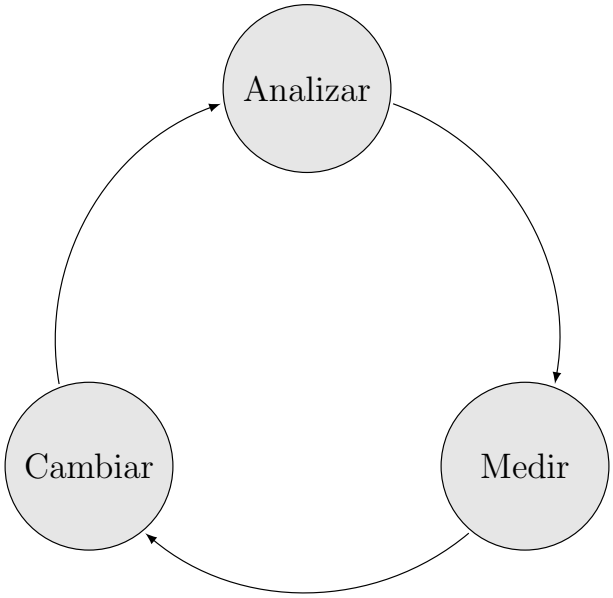

Figura 4.1: Ciclo de mejor de procesos

un evento es observado durante en el proceso. Por ejemplo, el número de pruebas software fallidas en el transcurso del proceso de validación.

Las dos primeras métricas pueden ser utilizadas para averiguar si los cambios introducidos en un proceso han mejorado su eficiencia. En cambio, la última métrica no tiene una traducción directa, si el cambio en el proceso permite descubrir un mayor número de defectos software no tiene porqué traducirse en un aumento de la calidad, deberán ser posteriores medidas las que corroboren o descarten el beneficio del cambio introducido.

Goal-Question-Metric

Saber qué información recopilar para llevar a cabo la mejora de procesos no es una tarea trivial. Basili et al. [63][64] propusieron a finales de los años ochenta el paradigma *objetivo-pregunta-métrica*, en inglés *goal-question-metric*, abreviado como "GQM". El paradigma GQM (figura 4.2) se utiliza como medio para responder tres preguntas cruciales en la mejora de procesos.

1. ¿Por qué se necesita la mejora de procesos?
2. ¿Qué información se necesita para identificar y evaluar las mejoras?
3. ¿Qué medidas se requieren tomar para facilitar esta información?

Estas preguntas se relacionan directamente con las abstracciones (objetivos, preguntas, métricas) en el paradigma GQM.

1. **Objetivo**. El objetivo es aquello que la organización persigue. No tiene por qué estar relacionado directamente con el proceso, pero si a como el proceso

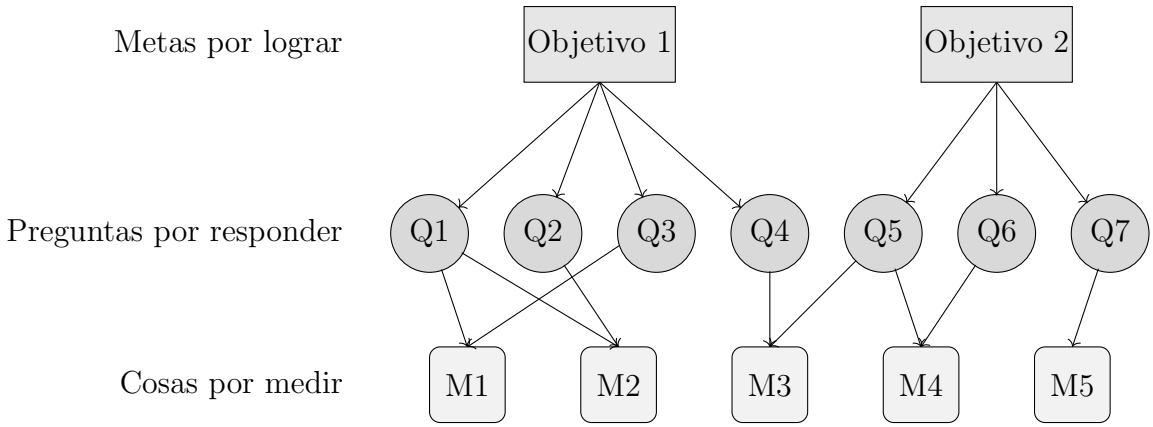

Figura 4.2: Paradigma GQM

afecta a la organización. Por ejemplo, si la organización se plantea eliminar el uso del papel en favor del soporte electrónico aquellos procesos que generen documentación deberán hacerlo en el nuevo soporte y se deberá definir cómo ser hará (tipo y codificación del archivo, lugar de almacenamiento, copias de seguridad, etc.).

2. **Preguntas**. Es el paso siguiente al objetivo. Buscan aflorar las zonas de incertidumbre relacionadas con el objetivo marcado. Normalmente, un objetivo se relacionará con un conjunto de preguntas a responder. Si el objetivo fijado es reducir el tiempo de un proceso cabrían preguntarse entonces ¿dónde están los cuellos de botella? ¿existen demasiadas reuniones? ¿son efectivas descubriendo defectos todas las pruebas realizadas?

3. **Métricas**. Constituyen las medidas que se deben recopilar para dar respuesta a las preguntas planteadas y para confirmar si una vez aplicadas las mejoras estas han surtido efecto. Algunas métricas útiles para responder a las preguntas planteas anteriormente podrían ser: porcentaje del tiempo que los desarrolladores se pasan en reuniones, número de casos de prueba que nunca detectan fallos, etc.

La ventaja de utilizar el paradigma GQM en la mejora de procesos es que permite separar las directrices organizativas (objetivos) de las propias del proceso (preguntas). Además, proporciona una base para decidir qué información recopilar y sugiere cómo se debe de tratar para dar respuesta a las preguntas planteadas.

La aproximación GQM ha sido desarrollada y combinada junto con los modelos de capacidad de madurez del Instituto de Ingeniería de Software [50] en la **metodología AMI** (*Analyze, Measure, Improve*) de mejora de procesos software. AMI propone un modelo por fases para llevar a cabo la mejora de procesos, donde las mediciones comienzas después de que la organización introduzca un estándar o norma a sus procesos. Pulford et al. [65] proporcionan una seria de guías a seguir para llevar a cabo una aplicación práctica del método AMI.

4.2.3. Análisis de Procesos

El análisis de procesos es el estudio de los procesos que permite comprender sus características clave y cómo esos procesos son llevados a cabo por las personas involucradas. El análisis está relacionado con la medida de los procesos en cuanto a que para poder recopilar información y luego realizar las medidas es necesario comprender qué se está midiendo e inevitablemente se debe entender cuál es el rol del atributo en cuestión dentro del proceso. Los objetivos que se persiguen en el análisis de proyectos son:

- Comprender las actividades involucradas en el proceso, y las relaciones que se establecen entre estas actividades.
- Comprender las relaciones entre las actividades del proceso y las medidas que se han hecho
- Relacionar el proceso bajo análisis y compararlo con el resto de procesos que se llevan a cabo en la organización o con la realización ideal del mismo.

Durante el análisis se persigue alcanzar una compresión de lo que sucede en el proceso y se buscan las posibles ineficiencias y problemas que puedan existir. También es interesante averiguar hasta qué punto el proceso es usado, cómo de importante o crítico resulta, cómo influye en el resto de procesos y cómo los requisitos y restricciones organizacionales le influyen a él. En la tabla 4.2 se detallan algunos aspectos a tener en cuenta a la hora de realizar el análisis de procesos.

Para realizar el análisis es habitual el empleo de las siguientes técnicas:

1. **Cuestionarios y entrevistas**. Se busca recopilar información de primera mano entrevistando a los ingenieros, jefes y directores involucrados en el proceso. Las respuestas al cuestionario formal son refinadas con una entrevista personal.
2. **Estudios etnográficos**. Se centran en los participantes del proceso, buscan entender el proceso de desarrollo como un proceso humano. Estos análisis revelan sutilezas y detalles imposibles de detectar mediante cuestionarios y entrevistas. Existen diversos trabajos sobre cómo llevar a cabo este tipo de estudios [66] [67] [68].

Cada uno de estos dos enfoques presenta ventajas e inconvenientes. El análisis basado en cuestionarios puede ser llevado a cabo razonablemente rápido si se identifican correctamente las preguntas adecuadas. Sin embargo, si las preguntas estas mal planteadas o son inapropiadas, el resultado puede ser una compresión incorrecta o poco aproximada a la realidad del proceso. Además, se corre el riesgo de que los entrevistados perciban el cuestionario como una evalución a su trabajo y sus respuestas estén enfocadas hacía lo que creen que sus superiores desean escuchar.

Las entrevistas permiten un marco más abierto, se parte de un guión establecido, pero el transcurso de la conversación puede derivar hacia aspectos no

Aspecto	Preguntas
Adopción y estandarización	¿El proceso esta estandarizado y documentado a lo largo de la organización? Si no lo esta, ¿significa esto que cualquier medición realizada solo es aplicada al proceso bajo análisis? Si el proceso no esta estandarizado, entonces los cambios puede no se aplicables a procesos similares en otros puntos de la organización.
Buenas prácticas	¿Existe alguna buena práctica de ingeniería de software no incluida en el proceso? ¿Por qué no lo esta? ¿La carencia de estas prácticas impacta en la calidad final del proyecto?
Restricciones organizacionales	¿Cuáles son loas limitaciones organizacionales que afectan al diseño y ejecución del proceso? Por ejemplo, si el proceso necesita manejar información clasificada o confidencial, existirán actividades que verifiquen esta información no sea incluida en informes que puedan salir de la organización.
Comunicación	¿Cómo se comunica la información en el proceso? Fallos en la comunicación son normalmente fuentes de problemas y cuellos de botella comunes en los procesos.
Introspección	¿Son los actores del proceso conscientes del mismo? ¿Pueden estos actores proponer cambios?
Aprendizaje	¿Cómo aprenden los nuevos integrantes los procesos en los que son participantes activos? ¿Tiene la organización algún programa estandarizado de aprendizaje?
Soporte de herramientas	¿Qué aspectos del proceso están apoyados por herramientas y cuáles no? ¿existen mejores herramientas disponibles?

Cuadro 4.2: Algunos aspectos del análisis de procesos.

contemplados en un primer momento. Normalmente este tipo de enfoques permite a los involucrados en el proceso expresarse mejor y de forma más abierta y natural que en el caso de los cuestionarios.

En casi todos los procesos, la gente involucrada tiende a realizar pequeños cambios locales para adaptar los procesos a circunstancias concretas. El análisis etnográfico persigue detectar estos cambios, y en general es más efectivo que cuestionarios y entrevistas a la hora de hacer esta detección. Sin embargo, es una tarea que requiere de un largo tiempo para llevarse a cabo y que puede extenderse meses. Se fundamenta en un análisis externo, sin interactuar en él, simplemente una observación detallada de todos sus fases y propiedades. Para realizar un análisis completo, se requiere observar todas las fases, por lo que en grandes proyectos esto podría conllevar un análisis de años, lo que lo hace en impracticable en muchas situaciones. A pesar de todo, este tipo de análisis suele ser útil cuando se desea profundizar no en la totalidad del proceso sino en una parte concreta del mismo.

Normalmente, el análisis comienza a partir del modelo del proceso, donde se define las actividades que lo componen con las entradas y salidas de dichas actividades. El modelo puede incluir también información acerca de los actores involucrados (el personal, los roles responsables de realizar las actividades, y los entregables críticos que se deben respetar). Pueden utilizarse diferentes herramientas para realizar el modelo, desde una notación informal hasta otra más formal como diagramas de actividad UML [25] o el modelo desde notación de procesos de negocio BPMN (Business Process Model and Notation) [69].

El modelo del proceso es una buena forma de centrar la atención en las actividades involucradas y en la información transferida entre ambas. Es importante tener en cuenta que el modelo no tiene porqué ser completo ni formal, su cometido es servir de base de discusión.

Al finalizar el análisis del proceso se deben tener un conocimiento profundo del proceso y una idea del potencial de mejora existente para el futuro junto con las limitaciones existentes a esas mejoras.

4.2.4. Proceso de cambio

El proceso de cambio supone introducir cambios y modificaciones al proceso existente. Los cambios pueden ser de muy diversos tipos: nuevas prácticas, métodos, herramientas, cambios en el orden de procesos y actividades, introducción o eliminación de entregables, nuevas formas de comunicación, introducción de nuevos roles y responsabilidades, etc.

Los cambios deben ser motivados y dirigidos por objetivos de mejora tales como *"aumentar la cobertura de las pruebas software al 90 %"*. Una vez que estos cambios necesarios se implementen, el proceso debe ser reevaluado y medido para comprobar la efectividad de los cambios.

Existen cinco etapas claves en el proceso de cambio de un proceso, tal y como se muestra en la figura 4.3.

- **Identificación de mejoras**. Esta etapa utiliza los resultados del proceso

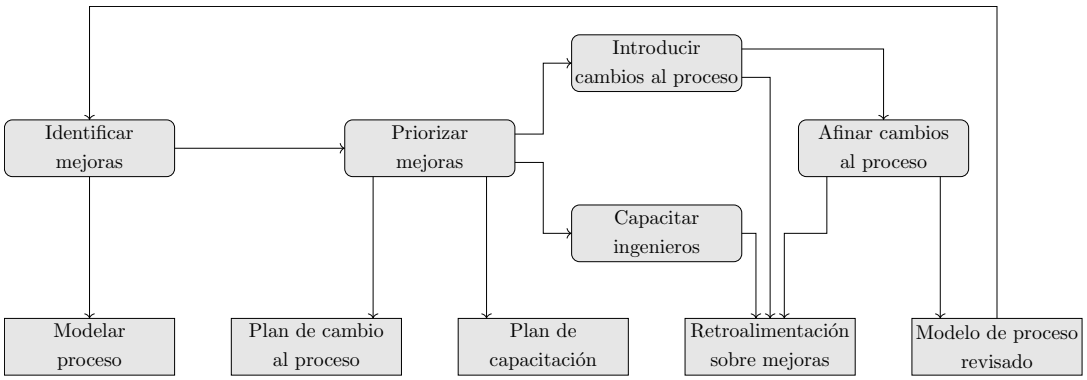

Figura 4.3: Proceso de cambio de procesos

de análisis para identificar maneras de atajar problemas de calidad, cuellos de botella en la planificación o ineficiencias en los costes. Se pueden proponer nuevos procesos, métodos y herramientas que afronten los problemas identificados. Por ejemplo, si una empresa detecta que tiene problemas en la identificación de requisitos podría introducir nuevas prácticas de recopilación de requisitos software o establecer algún método existente [70].

- **Priorización de las mejoras**. Cuando se detectan múltiples puntos de mejora, en ocasiones, resulta imposible implementarlos todos al mismo tiempo. Esta etapa busca ordenarlos de acuerdo al potencial impacto y coste que supondría implantar cada uno de ellos.
- **Introducción del cambio**. Etapa en la que se pone en funcionamiento los nuevos procedimientos, métodos y herramientas y se integra con las ya existentes.
- **Entrenamiento**. Esta etapa es clave para el éxito del cambio introducido, si no se entrena a los involucrados en el proceso cómo deben modificar su flujo de trabajo para adaptarse al cambio o no lo entienden el resultado del cambio puede no ser el esperado.
- **Ajuste**. Habitualmente el cambio introducido necesita ser alterado de uno u otra forma para minimizar pequeños problemas que suelen aparecer una vez implantado. Esta etapa puede durar varias semanas o meses.

No es recomendable introducir muchos cambios al mismo tiempo. Obviando el coste de formación de los cambios, introducir muchos cambios supone que la evaluación del resultado sea compleja o imposible. Si dos cambios tienen efectos opuestos, uno por ejemplo permite reducir a la mitad el tiempo en el que se realizan las pruebas software y el otro multiplica por dos ese tiempo, no se percibirá mejora alguna en el proceso resultante, y no se detectará cual es el proceso que realmente aporta una mejora real.

Además de la dificultad en la evaluación de la efectividad de los cambios introducidos, existen otros dos factos más a tener en cuenta.

1. **Resistencia al cambio**. No es raro encontrarse con reticencias a la hora de implantar un cambio por parte de las personas afectadas. En un primero

momento es normal recibir todo tipo de razonamientos acerca de porqué el cambio no funcionará o por qué no tiene sentido.

2. **Persistencia del cambio**. Aunque un cambio se introduzca con éxito en un principio no es extraño que pasado un tiempo este sea reconsiderado y descartado, regresando al punto de partida.

La resistencia al cambio puede proceder tanto de los directores y jefes de proyecto como del resto del personal. Directores y jefes de proyecto suelen resistirse a cambios en el proceso pues cualquier cambio conlleva consigo un riesgo asociado. Por otro lado, si el cambio necesita de un periodo largo de implantación para atajar una ineficiencia, el coste total a asumir puede no compensar el esfuerzo y por tanto ser descartado. El análisis entre el riesgo a asumir y el beneficio potencial del cambio es la principal razón por la que un director o jefe de proyecto rechace la implantación de un cambio en un proceso.

Los ingenieros y resto del personal tienden a rechazar cambios que afecten a sus competencias o perciban que menoscaban sus funciones y que en último caso les hagan prescindibles.

La tarea del director de proyecto debe ser la de velar por facilitar que todo el mundo implicado en el proyecto perciba los cambios en el proceso como algo natural y beneficioso, despejando dudas e involucrando a todos los miembros del proyecto afectados.

En ciertas ocasiones los cambios en los procesos vienen motivados gracias al impulso personal de algún miembro de la organización, lo que se conoce como "evangelizador". Este "evangelizador" es un firme convencido del cambio y hace todo lo posible para que el cambio llegue a introducirse. Sin embargo, cuando este "evangelizador" abandona el proyecto o la organización es frecuente que el cambio introducido sea sometido a reconsideración y la situación anterior reinstaurada. Esto es particularmente cierto cuando el cambio no ha sido establecido a lo largo de la organización y se percibe como una extravagancia de un determinado proyecto.

El problema de la persistencia del cambio es uno de los puntos en los que el marco CMMI pone toda su atención como veremos en las próximas secciones. La institucionalización de los cambios se presenta como una forma de conseguir que los cambios persistan más allá de un proyecto concreto o de una unidad de producción en particular.

4.3. Qué es CMMI

CMMI es una aproximación a la mejora de procesos a través del empleo de un conjunto de buenas prácticas utilizadas en la industria. Implementar CMMI puede ayudar a afrontar situaciones no deseadas por las organizaciones tales como, baja productividad, bajo rendimiento, costes no controlados, empeoramiento de la calidad y de la satisfacción del cliente.

El modelo CMMI describe aquello que la organización debería hacer, pero no el cómo lo debería hacer. El contexto empresarial de la organización debe ser quien

establezca como se implementan las prácticas propuestas. CMMI es un modelo de mejora no de conformidad al que las organizaciones deban adherirse. CMMI y sus métodos de evaluación permiten establecer una foto fija de la organización que refleje el estado de los procesos existentes. A partir de esta foto fija el marco de trabajo CMMI permite establecer una hoja de ruta para la mejora de estos procesos que reduzca las diferencias en el desempeño de los procesos entre la realidad medida y la ideal fijada por el modelo.

Es importante tener en cuenta qué no es CMMI para evitar un uso incorrecto del mismo:

- No es un estándar o norma preceptiva que deba cumplirse.
- No implica un trabajo extra dentro de la organización. Todas las organizaciones se guían por procesos para la realización de proyectos, fabricación de productos, la prestación de servicios o la realización de tareas diversas. La preocupación principal de CMMI es que estos procesos se hagan de manera formal de acuerdo a normas internas dentro de la organización que garanticen su repetitividad.
- No es una colección de procedimientos o procesos, sino que permite establecer procesos repetitivos y efectivos dentro de la organización.
- No es una "solución mágica" que pueda garantizar un aumento de la eficiencia en cualquier situación.

Es importante tener en cuenta que ningún modelo general podrá ser óptimo para todas las organizaciones. CMMI simplemente pretende ser un modelo que guíe a la organización a establecer un proceso o conjunto de procesos lo más óptimos posibles.

4.4. Modelo de Trabajo CMMI

CMMI fue desarrollado a partir del modelo CMM de mejorar de procesos por el Instituto de Ingeniería de Software conocido en inglés como Software Engineering Institute y abreviado por SEI, ubicado en la Universidad Carnegie Mellon. El SEI desarrolló un producto que contiene el modelo de mejora, los métodos de entrenamiento asociados al modelo y los métodos de evaluación.

Inicialmente CMMI se componía de un único modelo. Con el tiempo y gracias al éxito y dado que ciertas disciplinas tenían procesos únicos o con características singulares se desarrollaron tres modelos diferentes:

- **CMMI-DEV** centrado en el desarrollo de proyectos software
- **CMMI-ACQ** centrado en la adquisición o subcontratación de sistemas y servicios software.
- **CMMI-SVC**. centrado en la mejora de prestación de servicios software.

Estos tres modelos se complementan entre sí en términos del público al que se orientan. Existen varios componentes comunes a los tres modelos, incluyendo 16 áreas de interés denominadas "core process areas". Cada modelo CMMI también incluye áreas específicas a cada modelo.

4.4.1. CMMI-DEV

El modelo CMMI para desarrollo o CMMI-DEV fue el primero en ser desarrollado, y el modelo de interés para este TFM. El objetivo de CMMI-DEV es el de mejorar los procesos de desarrollo e ingeniería en aquellas organizaciones dedicadas al desarrollo de productos. Aunque CMMI está ligado a la industria software, el modelo CMMI-DEV puede ser usado en cualquier tipo de industria que produzca cualquier tipo de bien o producto.

4.4.2. CMMI-ACQ

El modelo CMMI para adquisición es utilizado para mejorar los procesos de gestión de proveedores en organizaciones con múltiples suministradores

El modelo de adquisición ayuda a las organizaciones a desarrollar requisitos, seleccionar proveedores y obtener servicios y productos.

CMMI-ACQ es ampliamente utilizado por organizaciones gubernamentales en los procesos de adquisición de productos y servicios y en la industria automotriz para la selección de proveedores.

4.4.3. CMMI-SVC

El último modelo en incorporarse al marco CMMI es CMMI-SVC orientado a servicios.

El objetivo de CMMI-SVC es el de ser utilizado como mejora de los procesos de gestión y suministro de servicios en organizaciones que desarrollen, gestionen o suministren servicios.

CMMI-SVC puede ser utilizado por cualquier organización que preste servicios. Entre las industrias que más han utilizado este modelo están las telecomunicaciones, empresas de suministro de energía, empresas de IT, call-centers etc.

4.5. Beneficio Potencial de CMMI

Las organizaciones difieren unas de otras en sus objetivos estratégicos, proyectos o servicios. Esto sucede aún incluso cuando se trate de organizaciones pertenecientes al mismo sector. Estas variaciones crean diferencias en como las organizaciones deben implementar la mejora de procesos mediante CMMI y de igual forma como deberán medir los resultados y el progreso obtenido durante y al finalizar la implantación de CMMI.

Categoría	Media de mejora
Coste	34 %
Plazos	50 %
Productividad	61 %
Calidad	48 %
Satisfacción de cliente	15 %
Retorno de inversión	4:1

Cuadro 4.3: Mejoras obtenidas por categorías.

Durante el proceso de mejora de procesos las organizaciones incurrirán en costes. Las inversiones realizadas en este proceso de mejora deben resultar en un aumento de la capacidad de los procesos y en la madurez de la organización. Estos beneficios pueden ser observados en el medio y largo plazo en forma de reducción de costes, plazos, productividad, calidad, satisfacción de usuario y de otras características.

El CMMI Institute ha realizado diversos estudios para evaluar el impacto que tiene la adopción de CMMI en las organizaciones. En la tabla 4.3 se muestran los datos obtenidos en cuanto a la mejora media obtenida en diferentes categorías.

4.6. Áreas de Proceso CMMI

Las áreas de proceso o process areas (PAs) son un conjunto de prácticas relacionadas en un área que, cuando se implementan de manera colectiva, satisfacen un conjunto de objetivos considerados importantes para hacer mejorar esa área. Las áreas de procesos tienen una estructura consistente a lo largo de los tres modelos. Cada una de estas áreas de proceso incluye:

- Objetivos considerados importantes para un área de negocio. Por ejemplo, el área de proceso de gestión de requisitos contiene todas las prácticas relacionadas con la gestión de requisitos, el seguimiento de los cambios de requisitos, y el control de los cambios.
- Prácticas.
- Material informativo.

En la figura 4.4 se muestra el diagrama con la estructura utilizada por cada área de proceso en CMMI. Cada PA comienza con una breve introducción que describe su propósito seguida de unas notas introductorias

CMMI incluye dos tipos de objetivos para cada área de proceso, por un lado, una serie de objetivos genéricos a todas las áreas de proceso y por otro un conjunto de áreas específicas a cada una de ellas. Cada objetivo, ya sea genérico o específico, tiene asociado un conjunto de subprácticas con el fin de proveer de una comprensión más profunda de la intención de dicha práctica. En el caso de las prácticas específicas, CMMI identifica ejemplos reales de productos o resultados de las prácticas para guiar mejor el objetivo del PA.

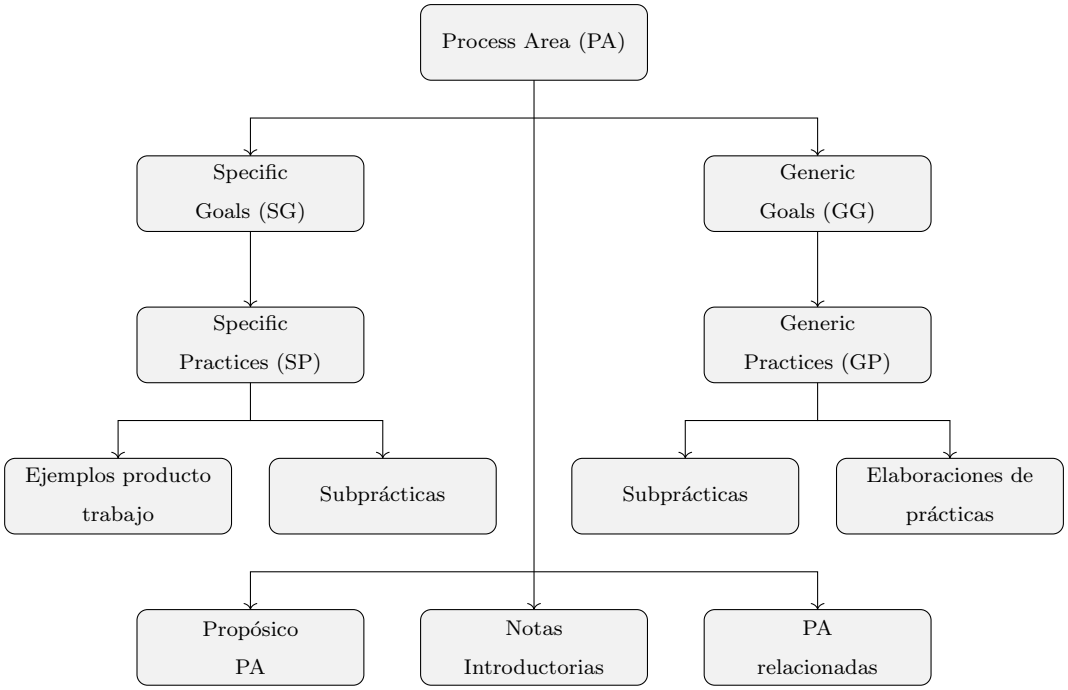

Figura 4.4: Esquema del área de proceso CMMI

Cada área de proceso comienza con estos componentes

- **Propósito**. Es un enunciado breve que resume el objetivo o propósito del área de proceso en cuestión. Debe ser una frase directa que transmita de forma clara y concisa la idea del PA.
- **Notas introductorias**. Consiste normalmente de varios párrafos que desarrollan la idea tras el propósito expuesto anteriormente. Es una forma de resumen ejecutivo del PA que describe la intención del PA e incluye ciertas guías de las prácticas involucradas den este PA.
- **Áreas de proceso relacionadas**. Es habitual que un PA se relacione con otras de una forma u otra, por tanto, se incluye una sección que liste aquellas otras PA que están fuertemente relacionadas. Normalmente no es un listado exhaustivo y solo se listan las PA con mayor relación.
- **Objetivo específico y Resumen de prácticas**. Cada objetivo y práctica asociada tienen un título y un enunciado asociado.

4.6.1. Objetivo Específico (SG)

El objetivo específico (SGs) es un componente del modelo requerido que describe las características únicas que deben estar presentes para satisfacer un área de proceso.

Por ejemplo, en el PA "gestión de requisitos" se establece un único objetivo específico: *"SG 1: Los requisitos deben ser gestionados y las inconsistencias de estos con el plan de proyecto identificadas"*.

Los objetivos específicos se identifican mediante el prefijo “SG” seguido por un número de forma secuencial.

4.6.2. Prácticas Específicas

Las prácticas específicas (SPs) son actividades consideradas importantes para conseguir los objetivos específicos asociados. Por ejemplo, para el caso anterior de gestión de requisitos: “SP 1.3: Gestionar los cambios de requisitos durante la ejecución del proyecto”.

Al igual que con los objetivos las prácticas se identifican mediante el prefijo “SP” seguidas de dos números separadas por un punto, el primer número hacer referencia al objetivo que cubre la práctica mientras que el segundo número es el número de la práctica asignado de manera consecutiva.

Las prácticas específicas suelen incluir dos tipos de componentes:

- **Una lista de ejemplos funcionales**. Esta lista representa los resultados que se pueden esperar encontrar en una organización que haya ejecutado estas prácticas. Los ejemplos no son elementos necesarios y no tienen por qué ser aplicables en otras organizaciones, tan solo muestran ejemplos típicos e ideas que puedan ayudar a poner en marcha la práctica específica a la que acompañan.
- **Subprácticas**. Proveen un grado de detalle mayor al descrito por la práctica y pueden contener un listado de pasos típicos para completar la práctica específica.

4.6.3. PA en los modelos CMMI

En los tres modelos CMMI (CMMI-DEV, CMMI-ACQ y CMMI-SVC) existen un total de 16 áreas de procesos centrales o core process areas comunes:

- Causal Analysis and Resolution (CAR)
- Configuration Management (CM)
- Decision Analysis and Resolution (DAR)
- Integrated Project Management (IPM)
- Measurement and Analysis (MA)
- Organizational Process Definition (OPD)
- Organizational Process Focus (OPF)
- Organizational Performance Management (OPM)
- Organizational Process Performance (OPP)

- Organizational Training (OT)
- Project Monitoring and Control (PMC)
- Project Planning (PP)
- Process and Product Quality Assurance (PPQA)
- Quantitative Project Management (QPM)
- Requirements Management (REQM)
- Risk Management (RSKM)

En CMMI-DEV existen además 5 áreas de procesos específicas:

- Product Integration (PI)
- Requirements Development (RD)
- Technical Solution (TS)
- Validation (VAL)
- Verification (VER)

Más otra área de procesa compartida con CMMI-SVC: SAM (Supplier Agreement Management). En CMMI-SVC además de la citada área de procesos compartida con CMMI-DEV existen otras 6 áreas de procesos específicas más otra adicional:

- Capacity and Availability Management (CAM)
- Incident Resolution and Prevention (IRP)
- Service Continuity (SCON)
- Service Delivery (SD)
- Service System Development (SSD)
- Service System Transition (SST)
- Strategic Service Management (STSM)

Por último, en CMMI-ACQ existen 6 áreas específicas:

- Acquisition Requirements Development (ARD)
- Solicitation and Supplier Agreement Development (SSAD)
- Agreement Management (AM)
- Acquisition Technical Management (ATM)
- Acquisition Verification (AVER)
- Acquisition Validation (AVAL)

4.6.4. Objetivos genéricos (GG)

Los objetivos genéricos describen características que demuestren la institucionalización del proceso en un área de proceso dada. Mientras que los procesos específicos aplicaban solo a un área de procesos, los objetivos genéricos aplican a todas las áreas.

El propósito de los objetivos genéricos es que una vez implementados estos permitan hacer sostenibles en el tiempo a los objetivos específicos. Sin la institucionalización de los objetivos genéricos hay una alta probabilidad de que las prácticas específicas se abandonen en momentos de alta exigencia de la organización.

Los objetivos genéricos se nombran en la misma forma que los objetivos específicos, comienzan con el prefijo GG seguido de un número.

4.6.5. Prácticas Genéricas (GPs)

Las prácticas genéricas son las actividades consideradas importantes o necesarias para lograr el objetivo genérico al que se asocian. Su carácter genérico implica que estas prácticas aparecerán en múltiples áreas de proceso. Un ejemplo de práctica genérica es: *"GP 2.5: Entrenar adecuadamente al personal involucrado en el proceso"*, y que estaría asociada con el área de proceso de entrenamiento de la organización (OT).

Las prácticas genéricas siguen el mismo esquema de numerado. Se emplea el prefijo GP seguido de dos números separados por un punto, donde el primer número hacer referencia al objetivo genérico asociado y el segundo número a la práctica genérica correspondiente.

Las prácticas genéricas están formadas típicamente por dos componentes:

- Elaboraciones de Prácticas Genéricas. Proporcionan una guía sobre cómo deben aplicarse la práctica genérica a un área de proceso específica. Son ejemplos de aplicación.

- Subprácticas.

4.6.6. Información Adicional

Además de proporcionar objetivos y prácticas tanto específicas como genéricas, los modelos CMMI también proporcionan otro tipo de información complementaría.

- **Ejemplos**. Normalmente en forma de lista de hitos, pasos, o elementos a tener en cuenta o que puede ser utilizados para facilitar la consecución de un objetivo o llevar a cabo de una práctica o subpráctica.

- **Referencias**. Al comienzo de cada PA se proporciona una lista de otras PA relacionadas. Esta lista no es exhaustiva, pero muestra las relaciones más importantes.

- **Notas**. Son aclaraciones, muy útiles para comprender mejor y en profundidad el objetivo, la práctica o subpráctica a la que hagan referencia. Suelen aclarar la intención que es esconde detrás y dar una visión de conjunto dentro del modelo del elemento al que hacen referencia.

4.7. Descripción Áreas de Proceso

A continuación, pasaremos a describir en detalle cada área de proceso perteneciente al modelo CMMI-DEV.

4.7.1. Requirements Management (REQM)

El propósito de la gestión de requisitos es identificar, asignar y hacer seguimiento de los requisitos necesarios para satisfacer los objetivos del proyecto. La gestión de requisitos es fundamental, una buena gestión de requisitos impacta muy positivamente en el proyecto ya que facilita el buen desempeño de los procesos y actividades siguientes. Es importante darse cuenta que las organizaciones no pueden gestionar a sus clientes, pero si puede gestionar los requisitos que reciben de ellos. Además, añadir rigor al proceso de gestión de requisitos genera una mejor compresión de los requisitos, facilita la gestión de los cambios de requisitos, y permite asegurar que el proyecto permanece consistente a los requisitos del cliente.

Existe un único objetivo específico para la gestión de requisitos, ver tabla 4.4.

Existen 5 prácticas específicas asociadas con el primer objetivo. SP 1.1 establece que la organización consiga alcanzar un conocimiento profundo de los requisitos solicitados en el proyecto. Una vez alcanzado este conocimiento, todos los involucrados deben tomar la responsabilidad de comprometerse con una parte de los requisitos. Los requisitos evolucionarán a lo largo del proyecto, bien por necesidades que irán surgiendo desde el punto de vista técnico o por necesidades surgidas por parte del cliente. Cuando los cambios suceden la organización debe asegurar que estos son gestionados (SP 1.3). Según vayan produciéndose cambios, la SP 1.4 se asegura que estos son traceables y que esta trazabilidad se mantenga a lo largo del proyecto. Los cambios de requisitos provocan a menudo desincronización dentro del proyecto, mientras que parte del equipo está trabajando con una versión de los requisitos (por ejemplo, el equipo de desarrollo) otro equipo puede estar trabajando con versiones anteriores (por ejemplo, el equipo de pruebas) originando ineficiencias. SP 1.5 se asegura que estas inconsistencias no existen.

4.7.2. Project Planning (PP)

El propósito de PP es "establecer y mantener planes que definan las actividades del proyecto". Incluye el desarrollo del plan de proyecto, identificar e involucrar a

SG1	**Gestionar Requisitos** Los requisitos son gestionados y las inconsistencias con el plan de proyecto y los productos de trabajo identificadas. **SP 1.1** Comprender los requisitos **SP 1.2** Obtener compromisos para los requisitos **SP 1.3** Gestionar los cambios de requisitos **SP 1.4** Mantener una trazabilidad bidireccional o de los requisitos **SP 1.5** Asegurar alineamiento entre el trabajo del proyecto y los requisitos

Cuadro 4.4: Área de proceso: Requirements Management (REQM)

todos los afectados relevantes por el proyecto (stakeholders), obtener compromiso para llevar a cabo dicho plan, y mantener el plan actualizado cuando este se vea modificado por cambios en los requisitos.

El plan de proyecto cubre varias actividades de gestión e ingeniería, así como de otros planes subordinados y compromisos con otros grupos.

Se establecen tres objetivos específicos para el PP, ver tabla 4.5.

Asociadas a los 3 objetivos específicos tenemos 4 prácticas asociadas para el SG 1, 7 para el SG 2 y 3 para el SG 3.

En SP 1.1 se establece la estructura de descomposición de trabajos EDT o también conocida en inglés como work breakdown structure (WBS) para poder así estimar el alcance del proyecto. La WBS proporciona una referencia y es utilizada como línea base para planificar, organizar y controlar el trabajo hecho en el proyecto. La WBS inicial es normalmente un documento de alto nivel que proporciona un marco inicial de trabajo que es completado con más detalle en tareas futuras a medida que el proyecto se desarrolla. SP 1.2 busca determinar cuánto trabajo debe ser empleado en cada paquete de trabajo identificado en la WBS, para ello se pueden utilizar diferentes métricas como número de páginas, funciones, tamaño o complejidad. La información obtenida en SP 1.2 es empleada en 1.4 para poder estimar los costes en función de la cantidad de trabajo, normalmente basándose en modelos existentes que tienen en cuenta parámetros históricos como número de actividades, tamaños, complejidades, etc. En SP 1.3 se definen las fases del proyecto dependiendo del número de requisitos, las estimaciones y la naturaleza del proyecto. En proyectos de gran tamaño es normal que las diferentes fases del proyecto sean a su vez divididas en sub-fases.

En cuanto al desarrollo del plan de proyecto, SP 2.1 establece el presupuesto y los plazos de acuerdo a la cantidad de trabajo y la WBS fijados durante el desarrollo de SG 1. Los riesgos del proyecto son identificados en SP 2.2. En SP 2.3 se desarrolla un plan para la gestión de los datos asociados al proyecto. Los

SG1	**Establecer estimadores** Se establecen una serie de estimadores de parámetros de proyecto que se mantendrán a lo largo de la vida del proyecto **SP 1.1** Estimar el alcance del proyecto **SP 1.2** Establecer una estimación del trabajo y las tareas a realizar **SP 1.3** Definir las fases del ciclo de vida del proyecto **SP 1.4** Estimar los costes
SG2	**Desarrollar plan de proyecto** Se establece un plan de proyecto como base para la gestión del proyecto **SP 2.1** Establecer el calendario y los costes del proyecto **SP 2.2** Identificar los riesgos **SP 2.3** Planificar la gestión de datos **SP 2.4** Planificar los recursos del proyecto **SP 2.5** Planificar las habilidades y conocimientos necesarios **SP 2.6** Planificar la participación de los involucrados **SP 2.7** Establecer el plan de proyecto
SG3	**Obtener compromisos con el plan de proyecto** Los compromisos con el proyecto son establecidos y mantenidos durante el ciclo de vida del proyecto **SP 3.1** Revisar los planes que puedan afectar al proyecto **SP 3.2** Poner de acuerdo el trabajo y los recursos compartidos **SP 3.3** Obtener compromiso de acuerdo para el plan de proyecto

Cuadro 4.5: Área de proceso: Project Planning (PP)

datos pueden aparecer de muchas maneras y medios. Se deben identificar los datos y fuentes de información claves para gestionar el proyecto de manera eficiente. En SP 2.4 se definen los recursos necesarios para llevar a cabo el proyecto. Los recursos incluyen, personal, herramientas, materiales e instalaciones. En ocasiones es necesario un conjunto de conocimientos o habilidades especiales para llevar a cabo el trabajo, SP 2.5 se asegura que estos son incluidos en el plan. En SP 2.6 se asegura que los involucrados son identificados correctamente de acuerdo a su relevancia en el proyecto, su rol, su necesidad y dónde deben participar. Todos estos elementos son empleados para crear un plan en SP 2.7, este plan es conocido como plan de proyecto y será utilizado para llevar a cabo la monitorización y control del proyecto.

Una vez el plan es documentado, se busca el compromiso de la organización para llevarlo a cabo. Antes de obtener el compromiso se debe verificar en SP 3.1, que el plan de proyecto no interfiere en otros planes, por ejemplo, si se el plan necesita de una instalación y esta debe ser compartida con otro proyecto. En SP 3.2 se establece que los recursos y el trabajo a realizar están equilibrados de forma realista y por tanto el proyecto es viable desde este punto de vista. Por último, SP 3.3 obtiene un compromiso por parte de todos los involucrados para llevar a cabo el proyecto.

4.7.3. Project Monitoring and Control (PMC)

El propósito de la monitorización y control es dotar un conocimiento acerca del progreso del proyecto de tal forma que se puedan tomar acciones correctivas en caso de que el proyecto se desvíe significativamente del plan inicial.

Se establecen dos objetivos específicos para llevar a cabo la monitorización y el control.

Los objetivos específicos asociadas a ambos objetivos específicos se detallan en la tabla 4.6. Las cinco primeras prácticas específicas del SG 1 hacen referencia a la monitorización de los elementos definidos en el plan de proyecto. Comienza en SP 1.1 monitorización el trabajo completado, el esfuerzo gastado, el presupuesto consumido, los plazos completados y los recursos empleados. SP 1.2 monitoriza que los compromisos asumidos por el proyecto son alcanzados. SP 1.3 monitoriza los riesgos. SP 1.4 monitoriza y verifica que el personal tiene acceso a los datos que necesita en todo momento y estos son gestionados apropiadamente. SP 1.5 se encarga de monitorizar que los compromisos asumidos por cada participante del proyecto se cumplen. Las dos últimas prácticas SP 1.6 y 1.7 tratan de las revisiones del proyecto y de los hitos cumplidos. El número y el tipo de estas revisiones depende enormemente del tipo de negocio y proyecto.

Al mismo tiempo que se monitoriza en el SG 1 el avance del proyecto se detectan desviaciones respecto al plan original del proyecto. En SG 2 se establecen las prácticas necesarias para corregir estas desviaciones. En SP 2.1 se analizan los problemas y se definen las acciones a tomar para corregir estos problemas. Las acciones correctivas se ponen en práctica en SP 2.2 y son seguidas y gestionadas hasta su finalización en SP 2.3. Es importante que las acciones correctivas tengan fechas límites para su finalización y que la monitorización se haga hasta el final.

SG1	**Monitorizar el proyecto contra su plan** Se monitoriza y compara el progreso y el desempeño del proyecto real y se compara contra el plan inicial **SP 1.1** Monitorizar los parámetros del proyecto o trabajo **SP 1.2** Monitorizar los compromisos **SP 1.3** Monitorizar riesgos **SP 1.4** Monitorizar gestión datos **SP 1.5** Monitorizar la participación de los involucrados **SP 1.6** Llevar a cabo revisiones del progreso **SP 1.7** Llevar a cabo revisiones de hitos
SG2	**Gestionar acciones correctivas** Las acciones correctivas son gestionadas hasta ser completadas cuando el proyecto o sus resultados se desvían significativamente del plan trazado **SP 2.1** Analizar problemas **SP 2.2** Llevar a cabo medidas correctivas **SP 2.3** Gestionar las medidas correctivas

Cuadro 4.6: Área de proceso: Project Monitoring and Control (PMC)

SG1	**Preparar la gestión de riesgos** Preparación para llevar a cabo la gestión de riesgos **SP 1.1** Determinar las categoría y fuentes de riesgo **SP 1.2** Definir parámetros de riesgo **SP 1.3** Establecer una estrategia de gestión de riesgo
SG2	**Identificar y analizar riesgos** Los riesgos son identificados y analizados para determinar su importancia **SP 2.1** Identificar riesgos **SP 2.2** Evaluar, categorizar y priorizar riesgos
SG3	**Mitigar riesgos** Los riesgos son abordados y mitigados de forma apropiada para reducir impactos adversos y lograr alcanzar los objetivos **SP 3.1** Desarrollar un plan de mitigación de riesgos **SP 3.2** Implementar el plan de mitigación de riesgos

Cuadro 4.7: Área de proceso: Risk Management (RSKM)

4.7.4. Risk Management (RSKM)

El objetivo de la gestión de requisitos es identificar problemas potenciales antes de que lleguen a producirse de forma que se puedan planear acciones y actividades que minimicen o palien estos riesgos. Existe una diferencia entre riesgos y problemas, un problema es algo que ya ha sucedido y que provoca un daño o pérdida, mientras que un riesgo es algo que potencialmente podría ocurrir y provocar un daño o pérdida. Las actividades de gestión de riesgos deben abordar aquellas casuísticas que puedan poner en peligro la consecución de objetivos críticos para lograr el éxito del proyecto. Una buena gestión de riesgos para por tener en cuenta tanto fuentes de riesgos externas e internas a la organización en cualquier eje importante del proyecto: plazo, coste y calidad. La detección temprana de riesgos suele ayudar a que, en caso de producirse un daño o pérdida motivada por el riesgo detectado, éste tenga menor impacto en términos de coste o plazo.

Se definen 3 objetivos específicos para este PA. Este patrón es seguido por otras PA a lo largo del marco de trabajo de CMMI, el primero objetivo suele establecer las herramientas y estrategias necesarias para ejecutar la tarea, mientras que las siguientes son objetivos de tareas.

Las prácticas específicas para los 3 objetivos se muestran en la tabla 4.7. En SP 1.1 la organización debe identificar fuentes y categorías de riesgos comunes. Esta lista de riesgos puede facilitar al proyecto un conocimiento ya adquirido de riesgos

comunes de forma que pueda servir de base en la evaluación de riesgos del proyecto. En SP 1.2 se establece una serie de parámetros de riesgo para poder evaluar, categorizar y priorizar los riesgos. Estos parámetros incluyen la probabilidad de riesgo, las consecuencias o severidad de los riesgos y los umbrales a los que el riesgo debe ser gestionado. La estrategia global de riesgos es finalmente definida en SP 1.3. Esta estrategia pasa por los métodos y herramientas utilizadas, las fuentes de riesgo específicas al proyecto, y cómo son los riesgos organizados, categorizados y priorizados.

En SG 2, las fuentes y categorías de SG 1 son empleadas para identificar riesgos en SP 2.1. Los riesgos deben ser identificados y descritos de forma comprensible y clara antes de que puedan ser analizados y gestionados adecuadamente. En SP 2.2, los parámetros definidos en SG 1 son utilizados para evaluar, categorizar y priorizar riesgos. El riesgo puede ser priorizado como alto, medio o bajo

La estrategia definida en SG 1 es utilizada para desarrollar planes de mitigación de riesgos en SP 3.1. Un componente crítico en la planificación de mitigación de riesgos es el desarrollo de caminos alternativos de acción, soluciones temporales o métodos de respaldo y caminos recomendados de acción para cada riesgo. El plan de mitigación de riesgos para un riesgo dado incluye técnicas y métodos utilizados para evitar, reducir y controlar la probabilidad de que el riesgo ocurra. Es importante tener en cuenta que la mitigación de riesgos aplica a una serie de riesgos seleccionados, no es viable el desarrollo de un plan de acción para todos los riesgos posibles. Los planes deben realizarse para aquellos riesgos con mayor probabilidad de ocurrir o de consecuencias más catastróficas. Los riesgos son monitorizados de manera periódica en SP 3.2 y los planes de mitigación implementados a lo largo de la vida del proyecto según las necesidades.

4.7.5. Supplier Agreement Management (SAM)

Esta PA es compartida por CMMI-DEV y CMMI-SVC y tiene como objetivo gestionar la adquisición de productos y servicios de un suministrador por parte de una organización.

El término suministrador se emplea para identificar a una organización externa al proyecto que desarrolle, produzca, pruebe y de soporte a un producto o servicio que es entregado a un cliente. El suministrador puede encontrarse dentro de la organización que patrocina el proyecto, pero ser ajeno al proyecto. Las actividades con el suministrador deben ser regidas por un acuerdo formal. Los acuerdos con proveedores podrán tomar forma de contratos, licencias o acuerdos de nivel o calidad de servicio.

Se definen dos objetivos específicos, tal y como se muestra en la tabla 4.8. Los suministradores son seleccionados en el SG 1 y los acuerdos establecidos y cerrados. En SG 2 el acuerdo se pone en práctica, es importante tener en cuenta que es un acuerdo de igual a igual y por tanto las exigencias son las mismas en ambas direcciones.

La tabla 4.8 muestra las prácticas especificas asociadas a SG 1 y SG2. En primer lugar, en SP 1.1 se determina qué tipo de adquisición se realizará para adquirir

SG1	**Establecer acuerdos con proveedores** Se establecen y mantienen acuerdos con proveedores **SP 1.1** Establecer el tipo de adquisición **SP 1.2** Seleccionar los proveedores **SP 1.3** Establecer acuerdos con proveedores
SG2	**Satisfacer los acuerdos con proveedores** Los acuerdos con proveedores deber ser satisfechos por ambas partes, el proveedor y quien ejecuta el proyecto **SP 2.1** Ejecutar el acuerdo con proveedores **SP 2.2** Aceptar el producto adquirido **SP 2.3** Asegurar la transacción de productos

Cuadro 4.8: Área de proceso: Supplier Agreement Management (SAM)

productos y servicios. El tipo de adquisición incluye el uso de componentes Commercial off-the-shelf (COTS) o de servicios de proveedores externos o internos. En SP 1.2 se seleccionan los proveedores de acuerdo a su capacidad para satisfacer los requisitos especificados para el producto o servicio a adquirir. Tras la selección del proveedor o proveedores se establecen acuerdos comerciales que establecen como ser efectúa la revisión, monitorización y los criterios de aceptación de los entregables del suministrador. Todo aquello no contemplado en el acuerdo comercial no debe ser contemplado que suceda en un futuro. Es importante incluir en el acuerdo todo lo necesario para poder gestionar al proveedor y recibir el producto o servicio esperado.

En la SP 2.1 se ejecutan los acuerdos establecidos en el SP 1.3. En SP 2.2 se comprueba que el producto o servicio cumple con los requisitos establecidos en el acuerdo antes de que sea entregado al cliente en el SP 2.3.

4.7.6. Configuration Management (CM)

Tiene como propósito establecer y mantener la integridad de los productos de trabajo utilizando la identificación de la configuración, el control de configuración, el registro del estado de configuración y las auditorías de configuración.

Existen tres objetivos específicos relacionados con CM, el primer objetivo SG 1 establece la línea base y el sistema de gestión de configuración que será utilizado en SG 2 y SG 3. SG 2 se asegura que los cambios a los productos de trabajo están controlados mientras que SG 3 requiere que las entradas de gestión de configuración sean guardadas y su integridad conservada.

Asociados a los objetivos tenemos una serie de prácticas específicas. SP 1.1 identifica los elementos de configuración que serán controlados. Algunos ejemplos

SG1	**Establecer líneas base** Las líneas base de los productos de trabajo son establecidas. **SP 1.1** Identificar los elementos de configuración **SP 1.2** Establecer y mantener un sistema de gestión de configuración **SP 1.3** Crear o liberar las líneas base
SG2	**Control de cambios de configuración** Los productos de trabajo bajo gestión de configuración son seguidos y controlados. **SP 2.1** Seguir las peticiones de cambio para los elementos de configuración **SP 2.2** Controlar los cambios a los elementos de configuración
SG3	**Mitigar riesgos** La integridad de las líneas base es establecida y mantenida. **SP 3.1** Establecer y mantener los registros que describen los elementos de configuración **SP 3.2** Realizar auditorías de configuración para mantener la integridad de las líneas base de configuración

Cuadro 4.9: Área de proceso: Configuration Management (CM)

de elementos de configuración a controlar son: documentos de diseño, descripción de servicios, materiales de formación, planes de pruebas, diseños técnicos y código fuente. Una vez los elementos a controlar son identificados, SP 1.2 establece un sistema para gestionar la configuración de esos elementos. Un sistema de gestión de configuración incluye los medios de almacenaje, procedimientos, y las herramientas para acceder al sistema. El sistema de gestión de configuración es utilizado para crear las líneas base de los elementos a controlar en SP 1.3. Una línea base representa una foto fija en el tiempo de la versión de configuración o de un conjunto de elementos de configuración.

La petición de cambios de configuración es analizada en SP 2.1 para determinar el impacto del cambio en el trabajo en curso, en los trabajos relacionados, en el presupuesto y en los plazos. Las peticiones de cambio son priorizadas, asignadas y seguidas hasta la finalización. SP 2.2 mantiene la línea base de configuración. Se establecen mecanismos para controlar qué se está cambiando, quien lo está cambiando, por qué se está cambiando y el estado del cambio.

Los registros de los cambios se establecen y gestionan en SP 3.1. En SP 3.2 se garantiza la fiabilidad de los registros, su consistencia y completitud.

4.7.7. Process and Product Quality Assurance (PPQA)

El propósito del Aseguramiento de la Calidad de Proceso y Producto es proporcionar a los diferentes equipos y a la gerencia una visión objetiva de los procesos y productos asociados. El objetivo fundamental de PPQA es garantizar que los procesos definidos están siendo respetados en la organización, así como poder detectar deficiencias en la forma de trabajar establecida. La objetividad es el elemento clave de PPQA. Aquellos miembros de la organización involucrados en PPQA deben separase de quienes se dedican al desarrollo del producto o llevan a cabo el proyecto. Se debe facilitar un medio de comunicación apropiado entre ambas áreas y los niveles de gestión para que los temas de no conformidad puedan ser tratados.

Existen dos objetivos específicos asociados con PPQA. SG 1 establece que tanto los procesos llevados a cabo como los resultados de los mismos deben ser objetivamente evaluados partiendo de las descripciones de proceso y los estándares establecidos. SG 2 aborda los temas de no conformidad detectados durante la validación y verificación del proceso.

Tanto SG 1 como SG 2 tienen asociadas cada uno dos prácticas específicas. SP 1.1 se concentra en la evaluación objetiva de los procesos llevados a cabo, verificando que cumplen con las descripciones de proceso asociadas, los procedimientos establecidos y los estándares utilizados. Esto va más allá de la simple inspección de los resultados obtenidos tras aplicar el proceso, tarea que se realiza en SP 1.2.

SG 2 se centra en la evaluación objetiva de los resultados. En SP 2.1 las no conformidades detectadas son comunicadas a los niveles adecuados de gestión para su planificación. Las no conformidades son monitorizadas para asegurar que estas son resueltas. Tanto las no conformidades como la solución o soluciones aplicadas se documentan en SP 2.2.

SG1	**Revisión de cumplimiento** La adherencia de los procesos realizados, y de los productos de trabajo asociados a las descripciones de proceso estándares y procedimientos aplicables es evaluada objetivamente **SP 1.1** Evaluación objetiva de procesos **SP 1.2** Evaluación objetiva de productos de trabajo
SG2	**Comunicación y seguimiento** Las no conformidades son seguidas y comunicadas objetivamente, y su resolución es asegurada **SP 2.1** Comunicar y resolver no conformidades **SP 2.2** Establecer registros

Cuadro 4.10: Área de proceso: Process and Product Quality Assurance (PPQA)

4.7.8. Measurement and Analysis (MA)

Tiene como propósito desarrollar y apoyar la capacidad de medición utilizada para poder dar soporte a las necesidades de información de la gerencia. Los directores tienen siempre la necesidad de contar con herramientas de medición que les permitan tomar decisiones en función del estado actual. La calidad de las mediciones es muy importante. Contar con mediciones inapropiadas puede causar la toma de decisiones que pongan en riesgo el éxito del proyecto. Las medidas no proporcionan respuestas en sí mismas, pero si pueden ayudar a que los directores se hagan las preguntas adecuadas acerca del estado y rumbo del proyecto. Establecer mediciones de los procesos permite establecer registros históricos que ayuden en evaluar la evolución de la madurez de la organización.

MA tiene dos objetivos específicos. En SG 1 se establecen los objetivos a medir y la especificación de las medidas y procedimientos a llevar a cabo para recopilar, almacenar y analizar dichas medidas. SG 2 lleva a cabo las tareas especificadas en SG 1.

Existen 4 prácticas específicas asociadas con SG 1. SP 1.1 espera que las necesidades de información establezcan unos objetivos a medir. En SP 1.2 se toman las medidas para lograr los objetivos especificados. Las medidas deben ser claramente definidas de forma que todo el mundo reporte el mismo tipo de información. Los métodos de recopilación de información son especificados en SP 1.3 con el fin de asegurar que todo el mundo sigue la misma metodología de recopilación y los datos son coherentes entre las diferentes fuentes de información. Por último, se establecen los procedimientos de análisis en SP 1.4 para evitar malinterpretaciones de los datos o un uso inadecuado de los mismos.

SG 2 cuenta también con 4 prácticas específicas. SP 2.1 recopila los datos de acuerdo con los procedimientos establecidos. Una vez recopilados los datos, se

SG1	**Alinear las actividades de medición y análisis** Los objetivos de medición y las actividades son alineadas con las necesidades y objetivos de información identificados **SP 1.1** Establecer objetivos de medida **SP 1.2** Especificar medidas **SP 1.3** Especificar procedimientos de recopilación y almacenamiento de datos **SP 1.4** Especificar procedimientos de análisis
SG2	**Proveer los resultados de las medidas** Se proporcionan los resultados de las medidas llevadas a cabo de acuerdo a las necesidades y objetivos identificados previamente **SP 2.1** Obtener los datos de la medición **SP 2.2** Analizar los datos de la medición **SP 2.3** Almacenar los datos y resultados **SP 2.4** Comunicar los resultados

Cuadro 4.11: Área de proceso: Measurement and Analysis (MA)

SG1	**Evaluación de alternativas** Las decisiones se basan en la evaluación de alternativas utilizando un criterio previamente establecido **SP 1.1** Establecer guías para el análisis de la decisión **SP 1.2** Establecer criterios de evaluación **SP 1.3** Identificar soluciones alternativas **SP 1.4** Seleccionar métodos de evaluación **SP 1.5** Evaluar soluciones alternativas **SP 1.6** Seleccionar soluciones

Cuadro 4.12: Área de proceso: Decision Analysis and Resolution (DAR)

analiza en SP 2.2 y se almacena en SP 2.3. Para finalizar, SP 2.4 comunica los resultados y conclusiones obtenidos tras el análisis a los involucrados apropiados para poder llevar a cabo tareas de decisión o actividades correctivas.

4.7.9. Decision Analysis and Resolution (DAR)

Tiene como propósito analizar las posibles decisiones utilizando un proceso de evaluación formal que evalúa alternativas identificadas contra los criterios establecidos. Una evaluación formal de procesos reduce la subjetividad de las decisiones y conduce a una mayor probabilidad de seleccionar la solución más apropiada para el mayor número de involucrados en el proyecto. La intención de DAR es que sirva de referente en la toma de las decisiones más importantes o críticas no en todas las situaciones. Una de las ventajas de DAR es que se deja registro de todas las decisiones, del qué y del porqué de la decisión.

Se establece un único objetivo específico. SG 1 establece la necesidad de explorar todas las alternativas posibles, evaluándolas de acuerdo a algún criterio previamente establecido.

DAR es una herramienta que puede ser aplicada en multitud de casos y actividades, incluyendo planificación, presupuestos, arquitectura, diseño, selección de suministradores, planificación de pruebas, logística y producción.

Las guías documentadas identifican cuando es necesario un proceso formal de decisión e incluyen los pasos a seguir en la toma de decisiones. SP 1.1 establece estas guías. SP 1.2 establece los criterios de evaluación a utilizar como base para la evaluación de soluciones alternativas. Estas soluciones alternativas son identificadas en SP 1.3. SP 1.4 selecciona los métodos a utilizar a ser utilizados en la evaluación de alternativas de acuerdo a los criterios de decisión. En SP 1.5 las alternativas son comparadas y evaluadas de acuerdo a los criterios y empleando los métodos de evaluación. Por último, en SP 1.6 se selecciona la solución o soluciones de acuerdo a las prácticas anteriores.

SG1	**Identificar mejoras al proceso** Las debilidades, fortalezas y oportunidades de mejora son identificadas de manera periódica o según las necesidades. **SP 1.1** Establecer necesidades de procesos **SP 1.2** Evaluar los procesos de la organización **SP 1.3** Identificar mejoras en los procesos
SG2	**Planificar e implementar acciones de mejora** Las acciones que acometan las mejoras a los procesos de la organización son planificadas e implementadas. **SP 2.1** Establecer planes de acción **SP 2.2** Implementar planes de acción
SG3	**Difundir las mejoras al proceso y recolectar las lecciones aprendidas** Los activos de proceso de la organización son difundidos en la organización y las experiencias relativas a procesos son incorporadas en los activos de proceso de la organización **SP 3.1** Desplegar los activos de procesos **SP 3.2** Difundir procesos estandarizados **SP 3.3** Monitorizar la implementación **SP 3.4** Incorporar las experiencias relacionadas con el proceso

Cuadro 4.13: Área de proceso: Organizational Process Focus (OPF)

4.7.10. Organizational Process Focus (OPF)

Tiene como propósito planificar, implementar y desplegar las mejoras de procesos de la organización, basadas en el entendimiento de las fortalezas y debilidades actuales de los procesos y de los activos de proceso de la organización. Este proceso establece la estrategia e infraestructura para el programa de mejora de procesos de la organización. Los activos de los procesos de la organización son utilizados para describir, implementar y mejorar los procesos de la organización. La organización debe asegurarse que las mejoras son puestas en marcha de forma efectiva y gradual.

Existen tres objetivos específicos para OPF. El primer objetivo, SG 1, pasa por identificar las fortalezas y debilidades de los procesos de la organización. Esta información debe ser utilizada para detectar puntos susceptibles de mejora. SG 2

establece un plan y responsabilidades para la mejora de procesos. La implementación, el despliegue y la monitorización de la mejora de procesos es responsabilidad de SG 3.

Existen 3 prácticas específicas relacionadas con SG 1. SP 1.1 requiere que la organización identifique sus necesidades y objetivos de mejora. Estas necesidades y objetivos deben establecerse de acuerdo a objetivos de negocio, necesidades y restricciones. Antes de realizar ninguna tarea de mejora es preciso conocer el estado de partida de la organización, esta tarea se realiza de forma periódica en SP 1.2, identificando también fortalezas y áreas de mejora. Las fortalezas deben ser potenciadas a lo largo de la organización mientras que las debilidades deben ser corregidas. Las mejoras a los procesos de la organización y a los activos de los procesos son identificados en SP 1.3.

Respecto al SG 2 se identifican dos prácticas específicas. En SP 2.1 se establecen planes de acción para mejorar procesos de forma específica, los cuales son aplicados y monitorizados hasta su conclusión en SP 2.2

Los activos de procesos son creados como parte de los esfuerzos de mejora de procesos de la organización. Una vez aprobados están listos para ser aplicados e implementados. La organización los desplegará en SP 3.1 en una actividad concreta. En 3.2 los proyectos y trabajos incorporan los nuevos procedimientos estandarizados. Con el tiempo se producirán actualizaciones y modificaciones que deberán ser también introducidas. La organización se asegura que la institucionalización y estandarización de los procesos es efectiva en SP 3.3. En SP 3.4 se incorporan las experiencias relacionadas con el proceso derivadas de la planificación y de la ejecución de los procesos, en los activos de proceso de la organización.

4.7.11. Organizational Process Definition (OPD)

El propósito de OPD es establecer y mantener un conjunto de ventajas del proceso organizacional y estándares de ambiente de trabajo. Una librería de ventajas del proceso de la organización es una colección de elementos utilizados por la gente y los proyectos de la organización. Esta colección de elementos incluye descripciones de proceso y elementos de proceso, descripciones de los modelos del ciclo de vida, guía para realizar un proceso, documentación relacionada al proceso y datos.

Las ventajas del proceso organizacional son utilizadas para realizar la implementación del proceso definido. Los estándares del ambiente de trabajo son utilizados para crear el ambiente de trabajo del proyecto. Un proceso estándar está compuesto de otros procesos o elementos de proceso. Un elemento de proceso es la principal unidad de definición del proceso y describe las actividades y tareas necesarias para realizar el trabajo. La arquitectura del proceso suministra reglas para conectar los elementos del proceso pertenecientes a un proceso estándar. El conjunto de procesos estándares de la organización puede incluir varias arquitecturas de proceso.

Existe un único objetivo en OPD. SG 1 requiere el establecimiento de un conjunto de activos de procesos organizacionales.

SG1	**Establecer activos de procesos organizacionales** Se establece y mantiene un conjunto de activos de procesos de la organización **SP 1.1** Establecer procesos estándar **SP 1.2** Establecer modelo de ciclo de vida **SP 1.3** Establecer guías y criterios para uso a medida **SP 1.4** Establecer el repositorio de medidas de la organización **SP 1.5** Establecer librería de activos de procesos **SP 1.6** Establecer estándares de entorno de trabajo **SP 1.7** Establecer guías y normas para los equipos de trabajo

Cuadro 4.14: Área de proceso: Organizational Process Definition (OPD)

Relacionados con el SG 1 existen 7 prácticas específicas. En SP 1.1 se establece el conjunto de procesos estándar existentes en la organización. Los elementos de los procesos estándar incluyen roles, estándares aplicables, procedimientos, métodos, herramientas, recursos, criterios de entrada y salida y medidas de proceso. En SP 1.2 la organización identifica que modelo de ciclo de vida es el más adecuado para ser utilizado en cada proceso. En SP 1.3 se recogen guía de uso para aquellas excepciones o necesidades particulares que puedan surgir a la hora de aplicar un procedimiento estandarizado. Existirán dos repositorios principales de información utilizados por el programa de procesos de la organización. El primero es el repositorio de medidas establecido en SP 1.4. El segundo es la librería de activos de procesos establecido en SP 1.5 que da soporte al aprendizaje de la organización y a la mejora de procesos facilitando y compartiendo buenas prácticas y lecciones aprendidas de anteriores proyectos. Elementos que puede formar parte de esta librería incluyen: políticas, descripciones de procesos, procedimientos, planes, materiales de aprendizaje y lecciones aprendidas. Los estándares del entorno de trabajo se establecen en SP 1.6 y permiten a la organización, los proyectos y trabajos beneficiarse de herramientas comunes, aprendizaje, mantenimiento y ahorros por compras centralizadas y en volumen. Por último, SP 1.7 se establecen guías para los equipos de trabajo.

4.7.12. Integrated Project Management (IPM)

El propósito de IPM es establecer y gestionar el proyecto y la involucración de todos los participantes relevantes dentro de un proceso integrado y definido ajustado al conjunto de procesos estándares de la organización. Gestionar el proyecto está relacionado con el proceso de definición de proyecto, el cual está hecho a medida de los procesos estándar de la organización.

Distinguimos dos objetivos dentro de IPM. SG 1 establece que los proyectos

SG1	**Utilizar procesos definidos** El proyecto es llevado a cabo utilizando procesos definidos y ajustados a la organización dentro del denominado conjunto de procesos estándar. **SP 1.1** Establecer el proceso de proyecto **SP 1.2** Utilizar los activos de proceso de la organización para planificar el proyecto **SP 1.3** Establecer el proyecto **SP 1.4** Plan integrado **SP 1.5** Gestionar el proyecto utilizando los planes integrados **SP 1.6** Establecer equipos **SP 1.7** Contribuir a los activos de procesos de la organización
SG2	**Coordinar y colaborar con los involucrados** Llevar a cabo una coordinación y colaboración entre la dirección del proyecto y el resto de involucrados. **SP 2.1** Gestionar la implicación de los involucrados **SP 2.2** Gestionar dependencias **SP 2.3** Resolver problemas de coordinación

Cuadro 4.15: Área de proceso: Integrated Project Management (IPM)

hagan uso de los activos de proceso estandarizados establecidos en la definición de procesos de la organización. SG 2 implica una aproximación proactiva a la hora de interactuar con el resto de involucrados en el proyecto.

Existen 7 prácticas específicas asociadas al primer objetivo. En SP 1.1 el proyecto utiliza un conjunto de procesos estándar de la organización y guías a medida para establecer un proceso definido para el proyecto en cuestión. El repositorio de medidas establecido en la definición de los procesos de la organización contiene información histórica de proyectos pasados que puede ser utilizado en SP 1.2 para estimar y dibujar el nuevo plan. A continuación, en SP 1.3 se crea un entorno de trabajo apropiado al proyecto, este entorno comprende las infraestructuras, herramientas y equipos que el personal necesita para llevar a cabo su trabajo de forma efectiva. Al igual que en la planificación de proyectos (PP) en donde se evaluaba como afecta el plan a otros en marcha, en SP 1.4 el proyecto integra otros planes para establecer un actuación conjunto de cara al resto de involucrados en el proyecto. En SP 1.5 se gestiona el proyecto empleando planes integrados de acuerdo al proceso definido. En SP 1.6 el proyecto es gestionado utilizando equipos de acuerdo a las reglas para estructurar, crear, formar y operar equipos. Durante el transcurso del proyecto, sobre todo en grandes proyectos, se podrán emplear di-

ferentes equipos con diferentes organizaciones de acuerdo a las necesidades. Por último, en SP 1.7 se recopila las experiencias y lecciones aprendidas para su documentación y almacena en el repositorio de medidas de la organización y en la librería de activos de la organización.

Para el SG 2 se identifican 3 prácticas específicas. SP 2.1 describe una aproximación más proactiva a la gestión de la implicación de los involucrados en el proyecto según el plan integrado y el proceso definido. En SP 2.2 se identifican las dependencias críticas con los involucrados en el proyecto. Y finalmente en SP 2.3 se resuelven cualquier tipo de problema de coordinación con los involucrados. Entre los problemas más habituales estarían las diferencias en la interpretación de requisitos y el diseño, retrasos, problemas a nivel de producto y la falta de recursos y personal disponible.

En la figura se han añadido las áreas específicas de la OPD para mostrar cómo se integran con IPM.

Un proceso definido comienza con activos de la organización que incluyen el conjunto de procesos estándar de la organización, estándares de entornos de trabajo, descripciones del modelo del ciclo de vida y reglas y guías de equipos. Estos activos son utilizados por un proyecto junto con las guías a medida y con la información del repositorio de medidas de la organización y la librería de activos para desarrollar la definición del proceso. Tras utilizar estos activos del proceso y sus procesos definidos, los proyectos suministran información de acuerdo a la experiencia acumulada a la librería de activos de procesos y al repositorio de medidas de la organización.

4.7.13. Organizational Training (OT)

El propósito de OT es el de desarrollar habilidades y conocimiento en el personal de manera que puedan llevar a cabo sus tareas de forma más eficaz y eficiente. Este PA trata de proveer de la formación necesaria para apoyar la visión estratégica de la organización y para cubrir las necesidades comunes en los proyectos realizados en la organización.

Se establecen dos objetivos para cubrir esta PA. El SG 1 establece la capacidad de la organización para ser capaz de ofrecer formación a sus integrantes, mientras que SG 2 es la encargada de llevar a cabo esa formación.

Asociado a SG 1 tenemos 4 prácticas específicas. SP 1.1 realiza un análisis de las necesidades de formación a largo plazo. Este análisis cubre típicamente las necesidades de formación en los siguientes uno a cinco años. En SP 1.2 se establece que grupo es responsable de qué formación. SP 1.3 establece el plan de formación para dotar a los miembros de la organización de conocimientos que les permitan incrementar su eficiencia y eficacia. Este plan cubre el medio y corto plazo y deber ser actualizado periódicamente para responder a los cambios que puedan aparecer. Por último, en SP 1.4 se establecen los métodos de formación, los materiales de formación, o la necesidad de buscarlos fuera de la organización.

SG 2 hace uso de las capacidades establecidas en SG 1 y cuenta con 3 prácticas específicas. SP 2.1 se asegura que la formación impartida se ajusta al plan de

SG1	**Establecer capacidad de formación** La capacidad de formación, que cubra los roles dentro de la organización, es establecida y mantenida **SP 1.1** Establecer necesidades estratégicas de formación **SP 1.2** Determinar que necesidades de formación son de responsabilidad de la organización. **SP 1.3** Establecer un plan de formación. **SP 1.4** Establecer la capacidad de formación
SG2	**Proveer formación** La formación de individuos es llevada a cabo de acuerdo a sus roles. **SP 2.1** Realizar la formación **SP 2.2** Establecer registros de formación **SP 2.3** Evaluar la efectividad de la formación

Cuadro 4.16: Área de proceso: Organizational Training (OT)

formación. SP 2.2 registra la formación impartida. Por último, SP 2.3 realiza las medidas oportunas para evaluar y si fuera el caso tomar medidas correctivas de los beneficios obtenidos de la aplicación de la formación.

4.7.14. Requirements Development (RD)

El propósito de RD es obtener, analizar y establecer requisitos de cliente, producto o de componentes de producto. Los requisitos deben también satisfacer las restricciones ocasionados por la elección de las soluciones de diseño.

Se establecen tres objetivos en RD. SG 1 hace referencia a la obtención y desarrollo de los requisitos de cliente. Estos requisitos de cliente son utilizados por SG 2 para crear los requisitos de producto y de componentes de producto. SG 3 analiza los requisitos y se asegura que estos son suficientes, necesarios y que cubren las necesidades operacionales.

Existen dos prácticas específicas relacionadas con SG 1. SP 1.1 implica la toma de forma proactiva de los requisitos del cliente, sus expectativas, restricciones y las interfaces existentes. Es habitual que los clientes no tengan una visión clara de lo que realmente necesitan y por tanto se debe colaborar de forma proactiva. Una aproximación temprana e iterativa de esta práctica conduce a una compresión más profunda y completa de las necesidades del cliente. SP 1.2 toma las necesidades obtenidas en el punto anterior y las utiliza para priorizar y desarrollar los requisitos de cliente. Cualquier información es relevante, y aquella información que no se pueda conseguir directamente a través del cliente debe intentar ser conseguido por

SG1	**Desarrollar requisitos de cliente** Se recopilan las necesidades de todos los involucrados en el proyecto, las expectativas, las restricciones y las interfaces. **SP 1.1** Obtener necesidades **SP 1.2** Transformar las necesidades de los involucrados en requisitos de cliente
SG2	**Desarrollo de requisitos de producto** Los requisitos de cliente son refinados y elaborados para desarrollar los requisitos del producto y sus componentes. **SP 2.1** Establecer requisitos de producto y componentes de producto **SP 2.2** Identificar requisitos de componentes **SP 2.3** Identificar requisitos de interfaces
SG3	**Analizar y validar requisitos** Los requisitos son analizados y validados. **SP 3.1** Establecer conceptos y escenarios operacionales **SP 3.2** Establecer una definición de la funcionalidad requerida y de los atributos de calidad **SP 3.3** Analizar requisitos **SP 3.4** Analizar requisitos para obtener un equilibrio **SP 3.5** Validar requisitos

Cuadro 4.17: Área de proceso: Requirements Development (RD)

otros medios. Además, los conflictos e inconsistencias que puedan surgir deben ser resueltas.

Respecto a SG 2 tenemos 3 prácticas. SP 2.1 parte los requisitos de cliente de SP 1.2 y los utiliza para crear los requisitos de producto y de componentes de producto. Mientas que los requisitos de cliente pueden ser de carácter no técnico los requisitos de producto si lo son y serán utilizados en el proceso de diseño. En SP 2.2 se identifican requisitos de más bajo nivel. En proyectos de alta complejidad es habitual encontrar varias capas de requisitos. Aunque las interfaces no dejan de ser componentes de cualquier producto, dado que son fuente habitual de procesos, CMMI establece una práctica independiente, SP 2.3, para su tratamiento.

SG 3 contiene 5 prácticas. SP 3.1 trata de establecer escenarios y conceptos operativos, son ideas de alto nivel de cómo debe comportarse el producto o servicio final. En SP 3.2 se establecen los requisitos funcionales y los atributos de calidad deseados. Esto puede incluir la documentación de la arquitectura funcional. SP 3.3 se asegura que los requisitos son suficientes para cubrir los objetivos del proyecto teniendo en cuenta los escenarios operaciones y los atributos de calidad establecidos. El cliente introducirá restricciones de diversa índole: coste, plazo, rendimiento funcionalidad, respuesta a imprevistos, mantenimiento y riesgos. SP 3.4 analiza los requisitos del cliente y las restricciones para buscar un equilibrio entre ambos. El último punto, SP 3.5, implica trabajar con el cliente para validar los requisitos para asegura que los trabajos a realizar y el producto final satisfacen todas las necesidades.

4.7.15. Technical Solution (TS)

El propósito de TS es seleccionar, diseñar e implementar soluciones a los requisitos. Las soluciones, diseños e implementaciones abarcan productos, componentes de producto y productos relacionados con el ciclo de vida de procesos tanto de forma individual o mediante combinaciones según convenga. Cuando nos referimos a producto también nos referimos a cualquier servicio.

Existen tres objetivos específicos. SG 1 requiere establecer la aproximación técnica a utilizar en el diseño y desarrollo del producto. SG 2 lleva a cabo el diseño y SG 3 la implementación y el desarrollo.

Relacionado con SG 1 tenemos 2 prácticas específicas. SP 1.1 establece que se deben buscar diferentes soluciones al problema y establecer un criterio que ayudar a seleccionar la más adecuada. Las soluciones alternativas son valoradas de acuerdo al criterio establecido en SP 1.2.

SG 2 abarca 4 prácticas específicas. En SP 2.1 se lleva a cabo el diseño del producto y de los componentes del producto. El diseño es utilizado durante todo el ciclo de vida del proyecto por los involucrados. Debe de ser de fácil compresión y permitir acomodar cambios con facilidad. SP 2.2 establece la necesidad de crear paquetes de datos técnicos. Un paquete técnico le proporciona al desarrollador una visión clara y comprensible del producto o de un componente del producto. Normalmente incluye la definición de la configuración del diseño, toda la información técnica aplicable, y una descripción de la alternativa técnica seleccionada para la

SG1	**Selección de las soluciones técnicas** Las soluciones elegidas son escogidas de entre un grupo de posibles soluciones alternativas **SP 1.1** Desarrollar soluciones alternativas y criterios de selección **SP 1.2** Seleccionar la solución
SG2	**Desarrollar el diseño** Se desarrollar los diseños del producto y sus componentes. **SP 2.1** Diseñar el producto o sus componentes **SP 2.2** Establecer un paquete de información técnica **SP 2.3** Diseñar las interfaces utilizando un criterio **SP 2.4** Realizar análisis de reutilización, compra o construcción
SG3	**Implementar el diseño del producto** El producto y sus componentes, así como la documentación asociada es implementada de acuerdo al diseño. **SP 3.1** Implementar el diseño **SP 3.2** Desarrollar la documentación

Cuadro 4.18: Área de proceso: Technical Solution (TS)

implementación. El diseño de las interfaces se realiza también de forma separa en SP 2.3. Por último, se establece que productos o componentes serán adquiridos, reutilizados o desarrollados desde cero o a partir de otros en SP 2.4.

Una vez el diseño es completado se implementa en SP 3.1. Ejemplos de implementación son: programación software, documentación de información y servicios, fabricación de partes y construcción de instalaciones. Esta práctica incluye peer reviews y unit-testing de los componentes del producto. SP 3.2 incluye las tareas de documentación, a menudo olvidadas, de instalación, operación y mantenimiento. La documentación debe tratarse como una pieza crítica de la solución técnica.

4.7.16. Product Integration (PI)

El propósito de PI es armar el producto a partir de los componentes del producto, asegurar que el producto armado se comporta adecuadamente y entregar el producto.

Se identifican 3 objetivos. En SG 1 se prepara la integración, en SG 2 se asegura la compatibilidad de la integración y en SG 3 se realiza.

La estrategia de integración se desarrolla en SP 1.1. Esta estrategia cubre la forma de recibir, ensamblar y evaluar los componentes que forman el producto. Antes de que la integración pueda comenzar, se debe establecer un entorno, esto se realiza en SP 1.2. El entorno requerido en cada paso de la integración del producto puede incluir equipos de pruebas, simuladores y equipos de medición. En SP 1.3 se establecen procedimientos para la integración junto con criterios para validar y entregar el producto final.

Al igual que en la etapa de diseño las interfaces tienen un objetivo propio en la etapa de integración. SP 2.1 revisa que las interfaces cumplan y cubran todas las especificaciones. Aplica tanto a las interfaces internas como externas. En SP 2.2 se pide la gestión de las interfaces. Esto incluye el mantenimiento de las interfaces a lo largo de la vida del proyecto teniendo muy en cuenta cómo pueden afectar cambios introducidos en la interfaz de un componente al resto.

Las prácticas específicas del SG 3 se orientan al ensamblado del producto y su entrega. En SP 3.1 se asegura que las partes a ensamblar cumplen con los requisitos de calidad y consistencia con las descripciones de las interfaces. El ensamblado tiene lugar en SP 3.2. La integración se lleva a cabo de acuerdo con la estrategia de integración, procedimientos y criterios establecidos en SG 1. Tras el ensamblado de los componentes, el producto es evaluado en SP 3.3. La última práctica, SP 3.4, constituye el empaquetado final del producto. Algunos productos tendrán especificado la forma en la que este empaquetado debe realizarse ante de ser entregado.

4.7.17. Verification (VER)

El propósito de VER es asegurar que el proyecto o producto cumple con sus especificaciones. La verificación se lleva a cabo a lo largo del ciclo de vida de desarrollo, comenzando con la verificación de los requisitos, pasando por la verificación

SG1	**Preparar integración** Se lleva a cabo los preparativos para la integración del producto. **SP 1.1** Establecer estrategia de integración **SP 1.2** Establecer el entorno de integración del producto **SP 1.3** Establecer los criterios y procedimientos de integración
SG2	**Asegurar compatibilidad de interfaces** Se asegura que las interfaces del producto tanto internas como externas son compatibles **SP 2.1** Revisar la descripción de las interfaces **SP 2.2** Gestionar las interfaces
SG3	**Ensamblar los componentes del producto y entregar el producto** Los componentes verificados del producto son ensamblados y el producto integrado, verificado, y validado es entregado. **SP 3.1** Confirmar la disposición de los componentes para la integración **SP 3.2** Ensamblar los componentes **SP 3.3** Evaluar los componentes ensamblados **SP 3.4** Empaquetar y entregar el producto

Cuadro 4.19: Área de proceso: Product Integration (PI)

SG1	**Preparar verificación** Se lleva a cabo los preparativos para la verificación. **SP 1.1** Seleccionar productos de trabajo para verificar **SP 1.2** Establecer entorno de verificación **SP 1.3** Establecer criterios y procedimientos de verificación
SG2	**Llevar a cabo revisiones** Se realizan peer-reviews (o revisiones por pares) **SP 2.1** Preparar peer-reviews **SP 2.2** Llevar a cabo peer-reviews **SP 2.3** Analizar la información de las peer-reviews
SG3	**Verificar** Se seleccionan productos para verificar los requisitos. **SP 3.1** Llevar a cabo verificaciones **SP 3.2** Analizar los resultados de las verificaciones

Cuadro 4.20: Área de proceso: Verification (VER)

de los productos y terminando con la verificación del producto final. La verificación en cada nivel del ciclo de vida aumenta las posibilidades de éxito.

Se establecen tres objetivos para la VER. El primero realiza una preparación, el segundo una revisión del primero y por último el tercero efectúa la verificación. Además, se establece un objetivo global para garantizar la institucionalización.

Asociados a SG 1 hay 3 prácticas específicas. En SP 1.1 se identifican los productos de trabajo susceptibles de ser verificados y los métodos a utilizar. Estas actividades de verificación ocurrirán a lo largo de todo el ciclo de vida del proyecto. No debe demorarse las actividades de verificación, cuanto antes se detecten fallos o no conformidades antes podrán ser resueltas y menor impacto se ocasionará en el coste o el plazo de ejecución. Los productos de trabajo a verificar son seleccionados de acuerdo a su peso en el proyecto y en los riegos que puedan originar. En SP 1.2 se establece el entorno necesario donde llevar a cabo la verificación. En SP 1.3 se establecen los criterios de verificación.

SG 2 cuenta también con otras tres prácticas específicas asociadas. En SP 2.1 se preparan las peer-reviews. Esta técnica es muy útil en los procesos de verificación. Ofrecen una gran oportunidad para aprender y compartir información. La preparación de las peer-reviews implica normalmente la identificación de los participantes, preparar y actualizar los materiales a utilizar durante la peer-review, tales como checklists y criterios de revisión y planificar las peer-reviews. En SP 2.2 productos de trabajo concretos son seleccionados y las peer-reviews llevadas a

SG1	**Preparar validación** Se lleva a cabo los preparativos para la validación **SP 1.1** Seleccionar productos para validar **SP 1.2** Establecer el entorno de validación **SP 1.3** Establecer los procedimientos y criterios de validación
SG2	**Validar** El producto o los componentes del producto son validados para asegurar que están preparados para su uso en el entorno real previsto. **SP 2.1** Realizar la validación **SP 2.2** Analizar los resultados de la validación

Cuadro 4.21: Área de proceso: Validación (VAL)

cabo sobre ellos. La información obtenida de estas reuniones es analizada en SP 2.3.

SG 3 involucra otras técnicas de verificación. La verificación implica a menudo probar, sin embargo, puede incluir también análisis, simulación y demostraciones. En SP 3.1 se emplean estos métodos para llevar a cabo la verificación. Para cada producto de trabajo sobre el que se realiza verificación SP 3.2 recopila y analiza la información obtenida.

4.7.18. Validación (VAL)

El propósito de la VAL es demostrar que el producto o los componentes del producto cumplen con las intenciones de uso cuando se sitúan en su entorno de uso.

Se diferencia de la verificación en cuanto a que la validación se centra en el uso del producto mientras que la verificación se centra en el cumplimiento de los requisitos.

La validación define dos objetivos específicos. SG 1 realiza los preparativos de la validación mientras que SG 2 se encarga de realizar la validación.

Ligado a SG 1 hay tres prácticas específicas. La validación comienza con la identificación de los productos a validar en SP 1.1. La selección se realizar en virtud de su relación con las necesidades del cliente final. En SP 1.2 se establece el entorno de validación. La validación, la verificación y la integración de producto pueden realizarse en el mismo entorno y compartir los mismos recursos. En SP 1.3 se seleccionan los criterios y procedimientos de validación que aseguren que el producto cumple con su objetivo de uso. Tras completar las actividades de SG 1, se realiza la validación en SP 2.1. Los resultados obtenidos en la validación son

analizados en el SP 2.2

4.7.19. Organizational Process Performance (OPP)

Tiene como propósito establecer y mantener una comprensión cuantitativa del rendimiento de los procesos seleccionados del conjunto de procesos estándar de la organización en apoyo al logro de los objetivos de calidad y de rendimiento de procesos, y proporcionar datos, líneas base y modelos de rendimiento de los procesos para gestionar cuantitativamente los proyectos de la organización.

Algunos términos ampliamente utilizados en OPP y otros PAs que debemos tener en cuenta son:

- **Rendimiento de proceso**. Es una medida de los resultados logrados al seguir el proceso. Caracterizado por:
  - Las medidas del proceso. Esto es, esfuerzo, tiempo del ciclo, eficiencia, etc.
  - Las medidas del producto o servicio. Por ejemplo, fiabilidad, densidad de defectos, tiempo de respuesta, etc.
- **Línea base de rendimiento de proceso**. Es una caracterización documentado del rendimiento del proceso y que puede incluir la tendencia central y su desviación. La línea base del rendimiento del proceso puede ser utilizada como medida para comprar el progreso actual contra el esperado.
- **Modelo de rendimiento de proceso**. Descripción de las relaciones entre los atributos medibles de uno o más procesos o de los productos de trabajo. Se desarrolla a partir de información histórica y se emplea para predecir el rendimiento futuro.

OPP solo tiene un único objetivo. SG 1 se encarga de establecer el modelo y las líneas base de rendimiento. Existe un paso previo que debe realizarse antes de establecer el modelo y las líneas base, es la identificación de los procesos susceptibles de ser medidos. La calidad y los objetivos de rendimiento de proceso son las medidas más útiles a la hora de seleccionar e identificar procesos para medir.

Asociado a este único objetivo hay 5 prácticas específicas asociadas. En SP 1.1 se establecen los objetivos de calidad y de rendimiento de proceso a partir de las necesidades de negocio. SP 1.2 selecciona los procesos o subprocesos dentro del conjunto de procesos estándar de la organización que se incluirán en los análisis de rendimiento de procesos de la organización y manteniendo la trazabilidad con los objetivos de negocio. SP 1.3 establece y mantiene definiciones de las medidas que serán incluidas en los análisis de rendimiento de procesos de la organización. SP 1.4 analiza el rendimiento de los procesos seleccionados, establece y mantiene las líneas base de rendimiento del proceso. Por último, SP 1.5 establece y mantiene los modelos de rendimiento de los procesos para el conjunto de procesos estándar de la organización.

SG1	**Establecer modelos y líneas base de rendimiento** Establece y mantiene las líneas base y los modelos, los cuales caracterizan la expectativa de rendimiento de los procesos para el conjunto de procesos estándar de la organización. **SP 1.1** Establecer objetivos de calidad y rendimiento de proceso **SP 1.2** Seleccionar procesos **SP 1.3** Establecer medidas de rendimiento de proceso **SP 1.4** Analizar el rendimiento de proceso y establecer las líneas base de rendimiento de proceso **SP 1.5** Establecer modelos de rendimiento de proceso

Cuadro 4.22: Área de proceso: Organizational Process Performance (OPP

4.7.20. Quantitative Project Management (QPM)

Tiene como propósito gestionar cuantitativamente el proyecto para alcanzar los objetivos establecidos de calidad y de rendimiento del proceso del proyecto.

QPM evoluciona la gestión de proyectos iniciada con las prácticas de PP y PMC y ampliadas con el proceso definido en IPM para introducir los conceptos de gestión estadística del proceso y la gestión cuantitativa del proceso para alcanzar los objetivos definidos en el proyecto. Requiere contar con información histórica suficiente de los indicadores del proceso recolectada por MA y controlada en OPD.

Se establecen dos objetivos para este PA. SG 1 describe cómo se prepara el proyecto para realizar gestión cuantitativa. SG 2 lleva a cabo las actividades definidas en SG 1.

SG 1 tiene asociadas cuatro prácticas específicas. En SP 1.1 se establecer y mantienen los objetivos de calidad y de rendimiento del proceso del proyecto. Estos se basan en los objetivos de calidad de lo organización. En SP 1.2 se utilizan la estadística y otras técnicas cuantitativas para componer el proceso definido que permite al proyecto alcanzar los objetivos de calidad y de rendimiento del proceso. Componer el proceso es complejo y por ello es habitual descomponer el proceso en subprocesos. Esta práctica implica identificar alternativas, realizar análisis cuantitativo del rendimiento y selecciona las alternativas que mejor encajen con las necesidades del proyecto. Los subprocesos que sean críticos para el éxito del proyecto suelen ser buenos candidatos para ser monitorizados y controlados utilizando técnicas estadísticas u otras técnicas cuantitativas. SP 1.3 Selecciona los subprocesos y atributos críticos para evaluar el rendimiento y que ayudan a alcanzar los objetivos de calidad y rendimiento del proceso del proyecto. SP 1.4 Selecciona las medidas y las técnicas analíticas a usarse en la gestión cuantitativa.

Respecto al segundo objetivo se identifican 3 prácticas específicas. SP 2.1 Monitoriza el rendimiento de los subprocesos seleccionados utilizando la estadística

SG1	**Preparación para la gestión cuantitativa del proyecto** Se lleva a cabo la preparación dela gestión cuantitativa del proyecto. **SP 1.1** Establecer los objetivos del proyecto **SP 1.2** Componer el proceso definido **SP 1.3** Seleccionar subprocesos y atributos **SP 1.4** Seleccionar medidas y técnicas analíticas
SG2	**Gestión cuantitativa del proyecto** El proyecto es gestionado cuantitativamente. **SP 2.1** Monitorizar el rendimiento de los subprocesos seleccionados **SP 2.2** Gestionar el rendimiento del proyecto **SP 2.3** Realizar el análisis de causa raíz

Cuadro 4.23: Área de proceso: Quantitative Project Management (QPM)

y otras técnicas cuantitativas. En SP2.2 se gestiona el proyecto utilizando la estadística y otras técnicas cuantitativas para determinar si los objetivos de calidad y rendimiento del proceso del proyecto son satisfechos. Por último, SP 2.3 realiza un análisis de causa raíz de los problemas seleccionados para atender las deficiencias en el logro de los objetivos de calidad y rendimiento del proceso del proyecto.

4.7.21. Causal Analysis and Resolution (CAR)

Tiene como propósito identificar las causas de los resultados seleccionados y tomar acción para mejorar la realización del proceso.

CAR, al igual que OPM, establece prácticas que permiten optimizar el proceso a nivel de proyecto o de la organización y requiere un entendimiento cuantitativo del proceso para poder ser efectivas.

Existen dos objetivos fijados en CAR. SG 1 está relacionada con la evaluación de las causas que originan los resultados que provocan variaciones como parte del proceso. SG 2 establece acciones específicas para mejorar los resultados y la capacidad del proceso en términos cuantitativos.

Asociado al objetivo uno tenemos dos prácticas específicas asociadas. En SP 1.1 se seleccionan los resultados que van a ser analizados. Esta selección implica recopilar información relevante y determinar qué resultados serán analizados en el futuro. Entre los métodos que pueden ser empleados para esta selección están: el análisis de Pareto, histogramas, diagramas de cajas y análisis de capacidad de procesos. Posteriormente, en SP 1.2 se realiza un análisis de causa raíz para

SG1	**Determinar las causas de los resultados seleccionados** Las causas raíz de los resultados seleccionados son determinadas sistemáticamente. **SP 1.1** Seleccionar resultados para analizar **SP 1.2** Analizar causas
SG2	**Atender las causas de los resultados seleccionados** Las causas raíz de los resultados seleccionados son atendidas sistemáticamente. **SP 2.1** Implementar las propuestas de acción **SP 2.2** Evaluar el efecto de las acciones implementadas **SP 2.3** Registrar la información de análisis causal

Cuadro 4.24: Área de proceso: Causal Analysis and Resolution (CAR)

determinar las causas subyacentes de los resultados seleccionados y de las acciones necesarias para atajar sus causas. Existen muchas herramientas que pueden ser utilizadas para este tipo de análisis, incluyendo análisis de causa-efecto, diagramas de Ishikawa, diagramas de Pareto o los cinco por qué.

El segundo objetivo tiene asociado tres prácticas específicas. Las propuestas de acción son desarrolladas a partir del análisis causa raíz en SP 2.1. Las propuestas de acción describen las tareas necesarias para atajar las causas raíz de los resultados analizados. Solo los cambios que tengan la capacidad de añadir valor significativo deben ser considerados para ser implementados de manera global. Una vez que el cambio en el proceso se implementa en los proyectos, los efectos de este cambio son evaluados para verificar que se ha mejora el rendimiento de proceso en SP 2.2. Por último, en SP 2.3 se registra toda la información obtenida del análisis. Este análisis es transmitido al resto de la organización en forma de propuesta de mejora. La organización analizará la propuesta en el área de proceso OPM.

4.7.22. Organizational Process Management (OPM)

El propósito de OPM es gestionar de manera proactiva el desempeño de la organización para alcanzar los objetivos de negocio. OPM analiza iterativamente información agregada de diversas fuentes con el fin de encontrar diferencias significativas entre el rendimiento observado y los objetivos de negocio. Además, busca seleccionar las mejoras que permitan reducir estas diferencias.

Existen tres objetivos específicos para mejorar el rendimiento de negocio. La organización no se centra en mejorar objetivos individuales, sino que persigue una

mejora global del rendimiento. La calidad y los objetivos de rendimiento de proceso se ajustan o se añaden nuevos objetivos en función de las necesidades de negocio. SG 1 emplea técnicas estadísticas y cuantitativas para identificar áreas donde el rendimiento cae por debajo de los objetivos de negocio. SG 2 requiere que la organización busque de forma proactiva e incremental ideas de mejora innovadoras. Aquellas ideas que potencialmente tengan un mayor impacto en el rendimiento son seleccionadas para ser puestas en práctica. SG 3 pone en marcha aquellos procesos mejorados, métodos y tecnologías en aquellos proyectos que puedan beneficiarse de las mejoras introducidas. Al mismo tiempo que se implantan los procesos mejorados se recopila información para asegurarse que los beneficios del cambio son reales.

Asociada al SG 1 hay 3 prácticas específicas. En SP 1.1 se utilizan los datos de rendimiento de la organización para evaluar si los objetivos de negocio son realistas y están alineados con las estrategias de negocio. Sin un conjunto bien definido de objetivos, la probabilidad de que las mejoras estén alineadas es menor. SP 1.2 utiliza el análisis de las líneas base de rendimiento de proceso, las simulaciones de proceso y modelos de rendimiento de proceso para evaluar la habilidad de la organización para conseguir sus objetivos de negocio. Basado en este último análisis, SP 1.3 identifica las áreas donde el rendimiento no está alcanzando el objetivo.

SG 2 cuenta con cuatro prácticas específicas. SP 2.1 se centra en recopilar mejoras provenientes de diversas fuentes. Las mejoras propuestas son analizadas en SP 2.2, estas se evalúan teniendo en cuenta su coste y su beneficio potencial. Basándose en esta evaluación, algunas mejoras son seleccionadas para su validación, implementación y posterior despliegue a lo largo de la organización. Los resultados del análisis se documentan en una propuesta de mejora. En SP 2.3 las mejoras se validan en el entorno de usuario. La validación se lleva a cabo en proyectos seleccionados que cumplan las características adecuadas en función del tipo de cambio a introducir. Tras el análisis y validación SP 2.4 selecciona finalmente las mejoras que serán implantadas en la organización.

Respecto al SG 3 existen tres prácticas específicas. En SP 3.1 se crea el plan de despliegue de cada mejora que va a ser implantada. Es necesario llegar a un compromiso entre estabilidad y cambio, la introducción de cambios con mucha frecuencia y poco impacto puede ser perjudicial. En SP 3.2 se gestiona el despliegue del plan de acuerdo a lo expuesto en el punto anterior. Los resultados de la implantación son evaluados en SP 3.3 empleando técnicas estadísticas y métodos cuantitativos.

4.7.23. Objetivos y Prácticas Genéricas (GG/GP)

Las metas genéricas (GG) y prácticas genéricas (GP) ayudan a institucionalizar en la organización las prácticas establecidas en cada una de las áreas de proceso, de manera que el proceso puede evolucionar desde un proceso "adhoc" hasta un proceso institucionalizado dependiendo del nivel que se quiera alcanzar.

Existen tres objetivos genéricos. Ambos objetivos están relacionados de forma que uno se fundamente en el otro como se muestra en la figura. El nivel de satis-

SG1	**Gestionar el rendimiento en la organización** El rendimiento del negocio de la organización es gestionado utilizando estadística y otras técnicas cuantitativas para comprender las deficiencias de rendimiento de proceso e identificar áreas de mejora de procesos. **SP 1.1** Mantener los objetivos de negocio **SP 1.2** Analizar la información del rendimiento del proceso **SP 1.3** Identificar áreas potenciales de mejora
SG2	**Gestionar el desempeño del negocio** Las mejoras son identificadas de manera proactiva, evaluadas usando técnicas estadísticas y otras técnicas cuantitativas y seleccionadas para su difusión basado en su contribución al cumplimiento de los objetivos de calidad y rendimiento del proceso. **SP 2.1** Obtener y categorizar las mejoras sugeridas **SP 2.2** Analizar las mejoras sugeridas **SP 2.3** Validar las mejoras seleccionadas **SP 2.4** Seleccionar e implementar las mejoras para difusión en la organización
SG3	**Seleccionar mejoras al proceso** Las mejoras medibles a los procesos y tecnologías de la organización son difundidas y evaluadas usando estadística y otras técnicas cuantitativas. **SP 3.1** Despliegue del plan **SP 3.2** Gestión de despliegue **SP 3.3** Evaluar los efectos de la mejora

Cuadro 4.25: Área de proceso: Organizational Process Management (OPM)

facción de los objetivos genéricos muestra el nivel de institucionalización de cada área de proceso CMMI

El GG 1 requiere que la organización lleve a cabo el trabajo necesario para producir productos de trabajo, en términos CMMI al proceso de le denomina **"performed process"**. Se realizan las prácticas específicas de las áreas de proceso. Sin embargo, la definición de proceso puede estar incompleta y la infraestructura necesaria para llevarlo a cabo ser deficiente. Como resultado no existe una garantía de que el proceso sea implementado consistentemente y duradero en el tiempo.

En GG 2 el proceso es institucionalizado. Se dice que el proceso está gestionado, típicamente CMMI se refiere a él como **"managed process"**, cuando está planificado y ejecutado de acuerdo a una política, emplea gente preparada con recursos suficientes, involucra a todos los afectados, esta monitorizado, controlado y revisado, y es evaluado de acuerdo a la descripción de su cometido.

GG 3 se fundamenta en el proceso gestionado de GG 2 para obtener un proceso definido o **"defined process"**. Para que un proceso sea definido este debe ajustarse al conjunto de estándares de la organización de acuerdo a una serie de guías predefinidas por la misma organización.

Las prácticas genéricas se relacionan con todas las áreas de proceso de CMMI

4.8. Representaciones CMMI

Las áreas de proceso de CMMI se pueden agrupar en torno a dos representaciones. Estas representaciones dan forma a cómo la organización lleva a cabo la mejora de procesos y las evaluaciones de los procesos. Las dos representaciones son: por etapas o continua. La motivación de que existan dos tipos de representaciones viene dada por el hecho de que las organizaciones pueden tener diferentes necesidades o prioridades a la hora de mejorar sus procesos.

4.8.1. Representación Continua

Esta representación ofrece un enfoque más flexible al permitir que las organizaciones aborden las áreas de proceso que consideren más necesarias. Una organización puede elegir mejorar una única área de proceso o un grupo de áreas de proceso que sean relevantes para las necesidades de negocio de la organización.

La tabla 4.27 muestra como se agrupan las áreas de proceso por categorías en el modelo CMMI-DEV.

La representación continua es utilizada para implementar mejoras de proceso de manera individual, esto es en áreas de proceso seleccionadas. La ventaja es que la organización elige el orden en el que efectuar las mejoras de acuerdo con sus necesidades. También permite a las organizaciones realizar las mejoras en diferentes PA a distinto nivel y ritmo. En la figura se muestran un ejemplo de esto último, diferentes PA de una organización con diferentes niveles de capacidad.

GG1	**Lograr objetivos específicos**
	GP 1.1 Llevar a cabo prácticas específicas
GG2	**Institucionalizar un proceso gestionado**
	GP 2.1 Establecer una política Organizativa
	GP 2.2 Planificar el proceso
	GP 2.3 Facilitar recursos
	GP 2.4 Asignar responsabilidades
	GP 2.5 Dar formación
	GP 2.6 Controlar los productos de trabajo
	GP 2.7 Identificar e involucrar a todos los afectados
	GP 2.8 Monitorizar y controlar el proceso
	GP 2.9 Evaluar la adhesión objetivamente
	GP 2.10 Revisar el estado con niveles de gestión superior
GG3	**Institucionalizar un proceso definido**
	GP 3.1 Establecer un proceso definido
	GP 3.2 Recopilar experiencias del proceso relacionadas

Cuadro 4.26: Procesos y prácticas genéricas (GG/GP)

Categoría	Área de Proceso
Gestión de Procesos	Organizational Process Definition
	Organizational Process Focus
	Organizational Performance Management
	Organizational Process Performance
	Organizational Training
Gestión de Proyecto	Integrated Project Management
	Project Monitoring and Control
	Project Planning
	Quantitative Project Management
	Requirements Management
	Risk Management
	Supplier Agreement Management
Ingeniería	Product Integration
	Requirements Development
	Technical Solution
	ValidationVerification
Soporte	Causal Analysis and Resolution
	Configuration Management
	Decision Analysis and Resolution
	Measurement and Analysis
	Process and Product Quality Assurance

Cuadro 4.27: Clasificación de áreas de proceso CMMI

4.8.2. Representación por Etapas

La representación por etapas ofrece un camino predefinido para llevar a cabo la mejora de procesos con CMMI. Cada paso en el camino requiere que se hayan completado los pasos anteriores y por tanto alcanzado un cierto nivel de madurez. Una ventaja de esta representación es que ofrece una calificación única de madurez para todas las áreas de proceso en lugar de múltiples calificaciones para cada PA.

Algunos beneficios de utilizar esta representación son:

- Proporciona una secuencia probada de pasos en el camino de la mejora de procesos. Avanzar de un nivel a otro garantiza que todos los objetivos hasta un cierto punto han sido alcanzados por la organización.
- Permite establecer comparaciones entre diferentes organizaciones en cuanto a su nivel de madurez de procesos.
- Proporciona una calificación única que recoge la evaluación de los resultados obtenidos de la mejora de procesos

Existen cinco niveles de madurez en esta representación, la tabla 4.28 muestras las PA afectadas en cada nivel.

4.8.3. Elección de representación CMMI

Las organizaciones que quieran utilizar CMMI como guía para la mejora de procesos deben elegir qué representación utilizar. Para ello deben pensar cómo van a implementar las áreas de proceso. Hay que tener en cuenta que el contenido es el mismo independientemente de la representación empleada. Deben ser las necesidades de negocio las que establezcan qué representación utilizar.

Las representaciones son a veces elegidas por las organizaciones en virtud del tipo de evaluación o **"appraisal"** que persigan. En una evaluación, las representaciones elegidas son utilizadas para monitorizar el progreso y el éxito de la implantación de las mejoras.

Las organizaciones que utilicen la representación por etapas llevarán a cabo evaluaciones que determinen si han conseguido cumplir con todas las áreas de proceso de un conjunto de PA predefinido con el fin de lograr un nivel de madurez.

En la representación continua, las evaluaciones tienen como objetivo determinar el nivel de capacidad de las áreas de proceso de forma individual.

4.9. Niveles CMMI

Los niveles son utilizados en CMMI como un camino que describe la evolución recomendada para que una organización mejore sus procesos. Los dos tipos de caminos de mejora que existen en CMMI están asociados a dos tipos de niveles

Nivel	Foco	Áreas de Proceso
5 Optimizado	Mejora continua de procesos	Causal Analysis and Resolution Organizational Performance Management
4 Gestionado cuantitativamente	Gestión cuantitativa	Organizational Process Performance Quantitative Project Management
3 Definido	Estandarización de procesos	Decision Analysis and Resolution Integrated Project Management Organizational Process Definition Organizational Process Focus Organizational Training Product Integration Requirements Development Risk Management Technical Solution Validation Verification
2 Gestionado	Gestión básica de proyectos	Configuration Management Measurement and Analysis Project Monitoring and Control Project Planning Process and Product Quality Assurance Requirements Management Supplier Agreement Management
1 Inicial		

Cuadro 4.28: Áreas de proceso asociadas a cada nivel de madurez del modelo por etapas CMMI.

y que corresponden a su vez a las dos representaciones existentes: continua y por etapas.

La aproximación por etapas es utilizada en las evaluaciones que determinan si la implementación de un área de proceso logra alcanzar o no un nivel de madurez predefinido. Los niveles de madurez muestran la evolución en el camino de la institucionalización y rendimiento de procesos.

La aproximación continua es utilizada en las evaluaciones que evalúan el nivel de capacidad de áreas de proceso individuales. Los niveles de capacidad muestran un camino de evolución basado en el grado de institucionalización y aptitud en el rendimiento de un área de proceso individual.

El concepto de nivel es el mismo independientemente de si se utiliza la representación continua o por etapas. Alcanzar un cierto nivel puede ser importante para las organizaciones, ya que es habitual que este constituya un requisito a la hora de obtener proyecto o de trabajar para otras empresas o instituciones.

4.9.1. Niveles de capacidad

Están relacionados con áreas de proceso individuales en vez de con un conjunto de áreas de proceso. El nivel de capacidad de un área de proceso se obtiene cuando se logra cuando se alcanzan hasta un cierto nivel todos los objetivos genéricos. Esto da una idea de la institucionalización del PA.

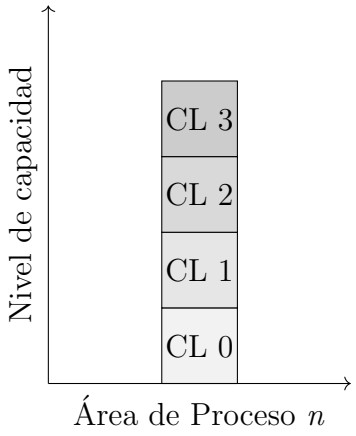

Figura 4.5: Niveles de capacidad CMMI

Existen cuatro niveles de capacidad numerados de 0 a 3. Estos niveles son acumulativos, es decir cuanto mayor es el nivel mayor es el grado de capacidad.

La capacidad se muestra mediante un gráfico de barras por cada área de proceso, como se muestra en la figura 4.5.

4.9.2. Niveles de madurez

El nivel de madurez representa un camino definido de pasos en la capacidad de los procesos. Permite medir los procesos contra un grupo predefinido de áreas de

proceso en el modelo CMMI. Cada nivel conseguido sirve como base para alcanzar el siguiente nivel, un nivel superior se basa en el inmediatamente anterior, y este en el anterior, y así de forma recursiva.

La clasificación del nivel de madurez se realiza en virtud de la satisfacción de todos los objetivos genéricos y específicos de las áreas de proceso agrupadas en el nivel correspondiente y en los niveles anteriores. Para conseguir un nivel 3 de madurez se deben cumplir todos los objetivos específicos de los niveles 1, 2 y 3.

Cada nivel de madurez sirve como capa y sustrato del nivel siguiente, tal y como se muestra en la figura 4.6. Los objetivos de cada nivel son:

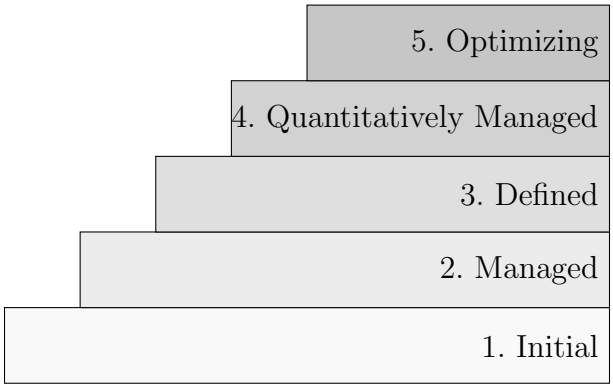

Figura 4.6: Niveles de madurez CMMI

- **Nivel 1**. El éxito depende del rendimiento de los individuos involucrados en el proyecto, los procesos son informales, ad-hoc, e impredecibles

- **Nivel 2**. Se realizan tareas básicas de gestión de proyecto y actividades para realizar un seguimiento y control básico del proyecto. Con esta gestión básica, la organización puede implementar aquellos elementos exitosos de la gestión en otros proyectos similares.

- **Nivel 3**. Se centra en la estandarización y despliegue a nivel organizacional de los procesos. Cada proyecto tiene un proceso de gestión que se adapta a los requisitos del mismo partiendo de un conjunto de procesos organizativos. Los activos y medidas de los procesos se deben recopilar y utilizar en futuras mejores de procesos.

- **Nivel 4**. Existe una metodología cuantitativa para controlar los subprocesos, esto es, las medidas de procesos recopiladas deben ser utilizadas en la gestión de procesos.

- **Nivel 5**. La organización utiliza las medidas de los procesos para dirigir su mejora de procesos. Las tendencias se analizan y los procesos modificados y adaptados a las necesidades de negocio.

4.10. Evaluación CMMI

Las evaluaciones o **"appraisals"** son el mecanismo utilizado para determinar el nivel de capacidad y madurez de una organización.

La figura muestra la estructura general de evaluación para las dos representaciones disponibles en CMMI.

Para la representación continua, se utiliza la satisfacción de objetivos genéricos para determinar el nivel de capacidad de las áreas de proceso individuales.

En la representación por etapas, la satisfacción de las áreas de proceso es empleada para determinar el nivel de madurez de la organización.

En las evaluaciones los componentes de CMMI son considerados de tres formas:

- **Requerido**. Describe lo que la organización debe lograr para satisfacer el área de proceso. Este logro debe ser implementado de manera visible en los procesos de la organización. La satisfacción de objetivos se utiliza en las evaluaciones como base para decidir si el área de proceso está satisfecha o no.
- **Esperado**. Mientras que los objetivos son considerados como requeridos, las prácticas se consideran esperables. Las prácticas representan las actividades que normalmente son esperables cuando la organización trata de satisfacer un objetivo genérico o específico. La manera en la que las organizaciones.
- **Informativo**.

Existen tres clases de evaluaciones estándar en CMMI para la mejora de procesos, también denominadas **SCAMPI**.

- **SCAMPI-A**. Es el método más riguroso y es el único método que obtiene como resultado una valoración o "rating".
- **SCAMPI-B**. Son métodos iniciales e incrementales, se utilizan como autoevaluación de los procesos.
- **SCAMPI-C**. Es un método que sirve para realizar una evaluación rápido y no rigurosa y que suele servir como aproximación a las organizaciones que deseen aplicar CMMI.

La tabla 4.29 muestra las variaciones entre los tres métodos de evaluación. El nivel de detalle aumento de C a B y de B a A. Normalmente las organizaciones emplean SCAMPI-C y SCAMPI-B como preparación para una auditoría basada en SCAMPI-A que les permite acreditarse con un nivel de madurez o capacidad CMMI.

Característica	Clase C	Clase B	Clase A
Cantidad de evidencias objetivas	Bajo	Medio	Alto
Puntuaciones generadas	No	No	Si
Recursos necesarios	Bajo	Medio	Alto
Tamaño del equipo	Pequeño	Mediano	Grande

Cuadro 4.29: Comparativa de los métodos de evaluación CMMI SCAMPI.

Capítulo 5

CMMI Ágil

Los métodos de desarrollo ágiles y las prácticas CMMI son a menudo percibidas como antagónicas y excluyentes entre sí. Sin embargo, esto está lejos de la realidad, CMMI y los métodos de desarrollo ágiles son compatibles y pueden por tanto usarse conjuntamente, es más, el correcto uso de ambos puede generar importantes sinergias que permitan aumentar la tasa de éxito de los proyectos software.

5.1. Percepción del CMMI Ágil

Si empezamos por analizar por qué CMMI y lo métodos ágiles se perciben como opuestos podemos dar dos razones que quizá nos ayuden a comprender esta situación [71]:

1. **Los primeros usuarios en adoptar CMMI y los métodos ágiles representan casos extremos**. Por un lado, quienes primero se iniciaron en CMMI normalmente fueron organizaciones desarrollando sistemas complejos, de carácter crítico, de tiempo real, de gran tamaño y con varios niveles organizativos de gestión. Por otro lado, las metodologías ágiles proliferaron en proyectos más pequeños, con equipos reducidos desarrollando software con requisitos cambiantes y poco estrictos. Estos dos extremos marcaron profundamente ambos campos.
2. **La información poco precisa sobre CMMI y métodos ágiles**, y el mal uso de ambos resultó en malinterpretaciones en ambos sentidos. Estas percepciones negativas que posicionaron a CMMI y los métodos ágiles como contrarios surgieron en gran medida a partir de los siguientes factores:
   a) Falta de uso o uso incorrecto.
   b) Falta de información precisa.
   c) Terminología confusa.
   d) Mejoras en el enfoque *Top-Down* contra el *Bottom-Up*

Comentaremos a continuación con un poco más de detalle los factores mencionados.

5.1.1. Falta de uso y uso incorrecto

Tanto CMM como CMMI son modelos y no estándares para mejorar el rendimiento de los procesos software dentro de las organizaciones. Sin embargo, con cierta frecuencia las organizaciones han utilizado las evaluaciones CMMI como un criterio para catalogar el rendimiento del negocio, de esta forma prima más la evaluación CMMI obtenida que los beneficios que se puedan obtener de ella.

Las primeras organizaciones que implantaron CMM y CMMI eran grandes organizaciones inmersas en grandes proyectos con complejos procesos, muchos de ámbito militar. Este origen dotó a CMM/CMMI de la fama de ser adecuado para este tipo de proyectos. Además, mucha de la literatura y libros escritos sobre la materia abordaban principalmente cómo poner en práctica CMM/CMMI en este tipo de proyectos y organizaciones.

5.1.2. Falta de información precisa

A pesar de que en los últimos años ha aumentado el número de publicaciones e información disponible acerca del uso de metodologías ágiles junto con CMMI, durante mucho tiempo esta información ha sido escasa y en ocasiones imprecisa.

Hasta 2005 no se había publicado información sobre experiencias de integración de metodologías ágiles y CMMI [72] exitosas. Algunos autores [71] señalan que el marcado carácter de ambos enfoques contribuyó a que durante mucho tiempo las experiencias de integración no fueran exitosas.

En 2008, el SEI publica un estado del arte sobre la integración de ambos enfoques [71]. A partir de entonces la idea del CMMI ágile ha ido cobrando más fuerza y el número de publicaciones y su calidad ha ido aumentando con el paso del tiempo.

5.1.3. Terminología confusa

Las metodologías ágiles, CMMI, los métodos de desarrollo tradicionales y los diferentes marcos de gestión de proyectos tienen todos ellos su propio vocabulario. Sin embargo, el solapamiento del vocabulario entre ellos resulta un problema a la hora de comunicar conceptos.

Por ejemplo, *technical data package* (TDP) es un término empleado por CMMI para referirse a una colección de documentos que describen el producto que está siendo desarrollado. Se emplea normalmente en desarrollos posteriores, operaciones, instalaciones, entrenamientos, soporte, resolución de dudas o mantenimiento del producto. El TDP puede incluir texto, diagramas, planos, diseños, especificaciones u otra información. Sin embargo, TDP tiene otro significado en el contexto en la adquisición de sistemas, donde se refiere a un entregable concreto que incluye una documentación específica.

Otro ejemplo de vocablo que puede inducir a confusiones es *predictable* o predecible, que tiene significados diferentes en el mundo ágil y el tradicional. Una de las ideas de la comunidad Agile es que los proyectos software no pueden ser

predecidos con precisión y por tanto "la perfección es el enemigo de lo suficientemente bueno, es mejor reaccionar y refactorizar a predecir". En cambio, CMMI utiliza el término *predictable* de manera más sutil. En el nivel 4 de CMMI, las predicciones se derivan a partir de la compresión de las variaciones esperadas a nivel de proceso junto con modelos estadísticos y probabilísticos que predicen los rangos de variación del proyecto. La predictibilidad no se consigue a partir de un plan minucioso que cubra todos los hitos del ciclo de vida del proyecto, ni siquiera CMMI espera o requiere este tipo de predictibilidad.

5.2. Combinando CMMI y Agile

CMMI y Agile son compatibles. A nivel de proyecto, CMMI se centra en un plano más abstracto, en *qué* hace el proyecto, no en que metodología de desarrollo que se emplea, mientras que los métodos Agile se centran en el *cómo* del desarrollo del proyecto. Por tanto, no existe ninguna razón que los haga incompatibles, pues trabajan de manera colaborativa en diferentes planos del proyecto con el fin de aumentar su tasa de éxito.

La combinación de ambos permite generar sinergias dentro de la gestión del proyecto. Hoy en día, existen muchas organizaciones que al tiempo que se adhieren a CMMI, han adoptado metodologías ágiles para el desarrollo software. De igual forma aquellas organizaciones acostumbradas a trabajar con metodologías ágiles, pueden adoptar CMMI para mejorar el proceso software.

5.2.1. Retos al usar métodos ágiles

El mayor reto que podemos identificar es el empleo de metodologías ágiles en el marco de grandes proyectos. En este tipo de proyectos es complicado mantener alineados a los diferentes grupos de desarrollo durante la ejecución del proyecto al mismo tiempo que se mantienen fieles a los valores y principios ágiles. Conseguir mantener el alineamiento a lo largo de un proyecto distribuido, esto es, un proyecto con diversos equipos (posiblemente separados geográficamente), requiere de alguien (o incluso otro equipo) o de un mecanismo encargado de mantener la coherencia de:

- las capacidades del sistema a desarrollar, incluyendo los requisitos no técnicos.
- alcance, calidad, plazo, costes y riesgos.
- arquitectura del producto o servicio.

Si no se logra la coherencia, la falta de alineamiento se puede hacer presente en diferentes frentes. Por ejemplo, un principio común en las metodologías ágiles, el *refactoring*, a menudo no escala bien cuando es necesario involucrar a diferentes equipos de desarrollo para llevarlo a cabo.

Las metodologías ágiles han conseguido escalar para poder ser utilizadas en grandes proyectos a través de diversas formas: *release planning*, *test planning*, *integración continua*, *despliegue continuo*, y utilizando nuevas arquitecturas como *microservicios* en lugar de componentes.

Desde el punto de vista de la ingeniería de sistemas, los siguientes puntos son conflictivos en grandes proyectos:

- visión global del proyecto.
- gestionar las necesidades de cada equipo.
- definir y mantener las interfaces y restricciones existentes entre equipos
- mantener una estrategia de integración, verificación y validación eficaz para todo el sistema.
- coordinar la gestión de riesgos a lo largo del proyecto.

Estos alineamientos y actividades de coordinación, necesarios en los proyectos más grandes, están descritos en las "prácticas de ingeniería de sistemas" que se pueden encontrar en CMMI bajo el epígrafe *Engineering, Risk Management, and Integrated Project Management process areas*. De esta forma CMMI provee una "red de seguridad" para grandes proyectos que ayuda a reducir los riesgos.

Los métodos ágiles son difíciles de aplicar como metodología principal de desarrollo en grandes proyectos. Sin embargo, recientemente se ha visto un aumento de su uso en este tipo de proyectos gracias a la introducción de nuevas formas ágiles. Una forma muy popular es añadir una nueva capa de gestión ágil que coordine al resto de equipos utilizando metodologías de desarrollo ágil, por ejemplo, una capa de gestión Scrum para coordinar al resto de equipos empleando Scrum, un "Scrum de Scrums". El empleo de metodologías ágiles en grandes proyectos es una tendencia que esta ganando terreno a medida que las organizaciones van ganando conocimiento en la aplicación de la metodología ágil.

La falta de guías y prácticas de uso de los métodos ágiles para su implementación y soporte a lo largo de una organización es otro problema a abordar. A pesar de que grandes organizaciones se han embarcado en la adopción de metodologías ágiles en todos sus proyectos, el nivel de las guías disponibles para su adopción es todavía escasa. Los métodos ágiles necesitan de un mecanismo que los haga adaptarse al contexto de la organización, a su filosofía y formas de trabajo. Es necesario introducir mecanismos que permitan medir y evaluar su impacto, y servir al mismo tiempo de guía para su mejora. CMMI puede ser ese mecanismo que permita que la inclusión de las metodologías ágiles en el desarrollo de proyectos software se convierta en un éxito.

Cabe decir que CMMI no es la única vía para conseguir implantar los métodos ágiles en grandes organizaciones de forma efectiva. Microsoft [73] ha desarrollado su propia técnica de mejora de procesos software. Si bien es cierto, que algunas de las ideas que ha implementado Microsoft son similares a las propuestas por CMMI, pero particularizadas para el entorno y las particularidades del negocio de Microsoft.

Otro reto a la hora de combinar CMMI y los métodos ágiles, y que por otra parte es común en casi cualquier actividad, es el factor humano. El factor humano se puede manifestar de dos formas:

- a través de una dirección poco convencida de los beneficios que se puedan obtener de la implantación de una nueva forma de trabajo.
- a través de un rechazo por parte de quienes deben modificar su forma de trabajo.

CMMI contempla estas situaciones y se ha diseñado para que la experiencia de cambio en los procesos de la organización afronte esta problemática. Por ejemplo, cuando se introduce un desarrollo ágil, es habitual que la dirección tema perder el control del proyecto.

5.2.2. Retos al usar CMMI

Al igual que con los métodos ágiles también existen retos a la hora de implantar CMMI. No existe ninguna aproximación o metodología que aborde todos los retos y situaciones que puedan surgir a lo largo de un proyecto software [74]. Simplemente por el hecho de que una organización obtenga un determinado nivel de madurez CMMI, no significa que los proyectos dentro de la organización vayan a ser todos exitosos. El nivel de madurez de una organización indica que la organización está preparada para gestionar aquello que puede hacer que un proyecto fracase hasta un cierto punto.

En ocasiones las organizaciones se ven en la necesidad de alcanzar un determinado nivel de madurez en un tiempo poco realista, bien porque lo necesitan para optar a conseguir un proyecto o bien por decisión de la dirección. En estas situaciones el objetivo deja de ser la mejora de procesos para ser el hecho de disponer de una acreditación del nivel de madurez. Así, se puede dar el caso de que se instauren normas, guías y métodos recogidos en CMMI pero que no sean los apropiados para la organización. En estos casos, la organización se acredita con un nivel de madurez, pero sin adoptar los procesos adecuados de acuerdo a sus necesidades reales. Por ejemplo, una organización de tamaño medio desarrollando software web adopta procesos de desarrollo, validación y verificación propios de una organización de gran tamaño desarrollando software crítico de tiempo real.

Las guías implantadas en una organización pueden no ser los suficientemente flexibles para un determinado proyecto, o para un proceso concreto del proyecto. Esta es una de las razones por las que el modelo de desarrollo ha evolucionado desde el modelo en cascada pasando por el iterativo o de espiral y llegando a los métodos ágiles. En un mundo tan cambiante como el actual, las organizaciones que implanten CMMI deben tener esto en cuenta y adaptarse a las situaciones de su entorno, cambiando si es preciso sus procesos a aquellos que mejor aborden los retos y problemas a los que se enfrentan.

5.3. Áreas de Proceso CMMI y SCRUM

En esta sección veremos cómo se pueden SCRUM en las áreas de proceso CMMI. De las 17 áreas de proceso descritas en CMMI-DEV no en todas podremos aplicar métodos ágiles, ni tampoco será posible aplicar metodologías de trabajo ágil a todas las prácticas asociadas a las áreas de proceso.

Se ha elegido SCRUM como metodología ágil por su amplia implantación y popularidad. Otros métodos ágiles tendrán otro tipo de relación con CMMI ya que no todos ellos cubren las mismas etapas del desarrollo.

5.3.1. Project Planning (PP) y SCRUM

Práctica Específica (PP)	**SCRUM**
SP 1.1 Estimar el alcance del proyecto.	Quedaría cubierto en la fase pregame de SCRUM en la que se define el backlog y los sprints. Ambos permiten realizar estimaciones del alcance.
SP 1.2 Establecer una estimación del trabajo y las tareas a realizar	SCRUM realiza una estimación inicial en el pregame que es refinada posteriormente en las reuniones de inicio de sprint.
SP 1.3 Definir las fases del ciclo de vida del proyecto	Queda cubierto por el propio ciclo de vida SCRUM
SP 1.4 Estimar los costes	Quedan estimados a nivel a nivel de producto en la reunión pregame y al comienzo de cada sprint. La estimación se realiza a partir de las historias de usuario.
SP 2.1 Establecer el calendario y los costes del proyecto SP 2.2 Identificar los riesgos	En el pregame se establecen hitos, (sprint goals), un calendario (sprints) y restricciones y presupuesto de acuerdo al backlog. Los riesgos se identifican mediante los impedimentos y registrados en la "lista de impedimentos". La identificación no es sistemática. Se hace de manera continua a lo largo del proyecto en las reuniones diarias. El ScrumMaster es el responsable de esta tarea.
SP 2.3 Planificar la gestión de datos	No abordado
SP 2.4 Planificar los recursos del proyecto	De nuevo en la fase de pregame se identifican las primeras necesidades del proyecto. En las reuniones diarias nuevas necesidades son identificadas y puestas en conocimiento del ScrumMaster.
SP 2.5 Planificar las habilidades y conocimientos necesarios	No abordado

Práctica Específica (PP)	SCRUM
SP 2.6 Planificar la participación de los involucrados	Normalmente participan en las reuniones que se realizan al comienzo y al finalizar cada sprint. Esta participación es responsabilidad del Scrum-Master.
SP 2.7 Establecer el plan de proyecto	Se podría decir que la visión de producto y el backlog permiten definir un plan de proyecto, aunque este plan es ciertamente difuso.
SP 3.1 Revisar los planes que puedan afectar al proyecto	Se llevan a cabo en las reuniones de planificación y en las reuniones retrospectivas.
SP 3.2 Poner de acuerdo el trabajo y los recursos compartidos	Ocurren durante las reuniones de planificación dado que el backlog es dinámico y por tanto los plazos son continuamente reestimados.
SP 3.3 Obtener compromiso de acuerdo para el plan de proyecto	Se obtiene de manera informar con las reuniones que se mantienen entre los involucrados del proyecto.

Cuadro 5.1: Mapeo de SCRUM en área de proceso (PP)

5.3.2. Project Monitoring and Control (PMC) y SCRUM

Práctica Específica (PMC)	SCRUM
SP 1.1 Monitorizar los parámetros del proyecto o trabajo	Reuniones diarias y retrospectivas.
SP 1.2 Monitorizar los compromisos	Reuniones diarias y retrospectivas
SP 1.3 Monitorizar riesgos	Reuniones diarias y retrospectivas
SP 1.4 Monitorizar gestión datos	No abordado
SP 1.5 Monitorizar la participación de los involucrados	Reuniones retrospectivas
SP 1.6 Llevar a cabo revisiones del progreso	Reuniones de revisión. Gráficas “burndown” y “burnup”
SP 1.7 Llevar a cabo revisiones de hitos	Reuniones de revisión
SP 2.1 Analizar problemas	Revisiones diarias y retrospectivas
SP 2.2 Llevar a cabo medidas correctivas	Reuniones de revisión
SP 2.3 Gestionar las medidas correctivas	Reuniones retrospectivas

Cuadro 5.2: Mapeo de SCRUM en área de proceso (PMC)

5.3.3. Requirements Management (REQM) y SCRUM

Práctica Específica (REQM)	**SCRUM**
SP 1.1 Comprender los requisitos	Historias de usuario de forma iterative
SP 1.2 Obtener compromisos para los requisitos	Reuniones de planificación. Backlogs
SP 1.3 Gestionar los cambios de requisitos	Reuniones de planificación y revisión
SP 1.4 Mantener una trazabilidad bidireccional o de los requisitos	Historias de usuario
SP 1.5 Asegurar alineamiento entre el trabajo del proyecto y los requisitos	Pregame y reuniones de planificación

Cuadro 5.3: Mapeo de SCRUM en área de proceso (PMC)

5.4. Ejemplo Práctico de CMMI Ágile

La literatura actual sobre CMMI ágil se centra en diferentes frentes de contacto entre ambos mundos. Tenemos publicaciones que muestran cómo satisfacer las áreas de proceso de CMMI mediante metodologías ágiles como SCRUM [75] [76] [77], o con diversas metodologías ágiles [78]. Existe también abundante literatura sobre los resultados de aplicar metodologías ágiles en organizaciones CMMI [79], o experiencias concretas de sectores diversos como el desarrollo de software de automoción [80], de defensa [81] o de desarrollo web [82] por citar algunos. Típicamente se aborda el nivel 2 de madurez CMMI, ya que las metodologías ágiles suelen cubrir las áreas de proceso de este nivel. No obstante, existe literatura acerca de cómo emplear técnicas ágiles, o modificaciones de metodologías ágiles, para conseguir niveles superiores de madurez [83] [84].

En todos estos casos se suele evaluar la mejora en el proceso software a partir de la implantación del CMMI ágil de acuerdo a una serie de parámetros observables. Sin embargo, a juicio del autor, no se ha estudiado en profundidad y de manera detallada cómo llevar a cabo la puesta en marcha del CMMI ágil. En esta sección se analizará las técnicas concretas necesarias para que una organización implante un proceso de desarrollo software ágil capaz de satisfacer los requisitos CMMI.

5.4.1. Escenario base

Vamos a considerar el desarrollo de una aplicación compleja en la que es necesario involucrar a diversos equipos de desarrollo responsables de diferentes subsistemas que deben colaborar entre sí.

En este caso vamos a plantear el desarrollo de un sistema de captura, envío, pro-

cesado y almacenamiento de información. Los datos serán capturados por sensores que se encontrarán ubicados en cualquier punto geográfico reportando medidas a un servidor central. El servidor central podrá recibir información en cualquier momento de diferentes sensores, procediendo al procesado de la información, la toma de decisiones en virtud de los resultados obtenidos tras el procesado y el almacenamiento de toda la información para posteriores análisis o consultas. Para implementar el sistema lo descompondremos en subsistemas más simples, de tal forma que diferentes equipos de desarrollo puedan participar en paralelo en la implementación. De esta forma los subsistemas que constituyen la solución completa son:

- **Interfaz de usuario**. Podría ser una interfaz de escritorio, web o aplicación móvil, en este caso hemos seleccionado que sea una aplicación web. Las aplicaciones web suelen dividirse a su vez en dos partes:
  - **Frontend**. Correspondiente a la parte gráfica.
  - **Backend**. Correspondiente a la parte lógica encargada de interpretar las acciones del usuario.
- **Módulo de procesado**. Correspondiente a la parte lógica encargada de interpretar las acciones del usuario
- **Fuente de datos**. Software ubicado en algún dispositivo remoto que produce información.

El proceso de desarrollo software se llevará a cabo mediante SCRUM. En este caso ya que el desarrollo está dividido en diferentes equipos tendremos lo que se conoce como un SCRUM de SCRUMs. Se llevarán a cabo reuniones periódicas, entre dos y tres a la semana, con una duración de entre 30 y 60 minutos a las que acudirá un miembro de cada equipo para tratar aquellos temas que sean comunes. Normalmente en las reuniones diarias SCRUM no se tratan de buscar soluciones, sin embargo, en este tipo de reuniones si es que se deben establecer soluciones concretas a los problemas detectados. Esto es así para que la información fluya rápidamente entre todos los equipos y poder detectar posibles incompatibilidades entre subsistemas.

5.4.2. Implementación CMMI y SCRUM: Gestión del proyecto

A la hora de gestionar proyectos se hace indispensable el uso de herramientas software que permiten al director del proyecto gestionar los diferentes procesos involucrados. Tradicionalmente, herramientas como *Microsoft Project* u *Oracle Primavera* han sido empleadas para la gestión de todo tipo de proyectos, sin embargo, en los últimos años han aparecido otras herramientas para la gestión de proyectos ágiles que se adaptan mejor a nuestro escenario. Entre el amplio abanico de candidatos disponibles, dos herramientas destacan sobre el resto

- **Asana**. Es simple de utilizar y se puede integrar con otras herramientas de desarrollo software en la nube como GitHub, Slack, Okta, Google Drive, etc.
- **Atlassian JIRA**. Forma parte del conjunto de aplicaciones de Atlassian (muy centrados en el apoyo al desarrollo de software), que pueden ser utilizadas tanto en la nube como en datacenters privados. JIRA es más compleja que Asana y requiere de un mayor esfuerzo tanto por parte del encargado del proyecto como del resto de participantes.

En nuestro caso, emplearemos JIRA junto con Microsoft Project, buscando de alguna manera reflejar también a nivel de software las sinergias que aparecen entre la visión clásica de gestión de proyectos y el enfoque ágil. Microsoft Project nos permitirá llevar el seguimiento de más alto nivel del proyecto, desde las primeras fases de creación de equipos, definición de objetivos de muy alto nivel, elaboración del presupuesto, etc, pasando por los principales hitos del desarrollo, hasta el cierre y entrega del proyecto al cliente. Cada paquete de trabajo de la WBS se gestionará a través de JIRA de forma ágil. Gracias a la posibilidad de conectar Project y JIRA podremos de forma automática sintetizar toda la información JIRA correspondiente a un paquete de trabajo y enviarlo a Project para que actualice el estado del proyecto. Esta forma de trabajo encajará perfectamente en el equilibrio que perseguimos entre CMMI y SCRUM.

JIRA gira en torno al concepto de *issue*. La *issue* permite modelar diferentes conceptos sujetos a un ciclo de vida, por ejemplo, una petición de cambio, un bug, una reunión, un informe, etc. JIRA proporciona una serie de *issues* por defecto y permite al mismo tiempo crear nuevas *issues* con su lógica y ciclo de vida asociado. De esta forma podremos crear *issues* para cumplir con las áreas de proceso de CMMI y con la metodología SCRUM. De esta forma definiremos la siguientes *issues*:

- **Story (US)**. Describirá una historia de usuario que contendrá un requisito a implementar descrito en unas frases breves empleando un lenguaje no técnico. Dentro de esta historia de usuario se irán creando sub tareas que representarán elemento de trabajo necesarios para satisfacer la historia de usuario.
- **Epic (EP)**. Representa una porción de trabajo extensa. Se puede entender como una historia de usuario más amplia de la habitual y que podría descomponerse en otras más simples. Podría darse el caso de que esta *issue* afecte a más de un proyecto o que fuesen necesarios más de un sprint para completarla.
- **Technical Task (TT)**. Las tareas técnicas son aquellos trabajos necesarios para el proyecto pero que no se corresponden con ninguna funcionalidad del producto o servicio sobre el que se esté trabajando. Un ejemplo podría ser la preparación de un entorno de desarrollo, la configuración de los equipos de red, la instalación de un software determinado, etc.
- **Change Request (CR)**. Permite realizar un seguimiento de una petición de cambio.

- **Risk (RSK)**. Esta *issue* modelará los riesgos potenciales del proyecto y permitirá hacer seguimiento de los mismos. Un ejemplo de aplicación es cuando se procede a realizar un *merge* entre dos ramas de desarrollo del repositorio de código fuente.
- **Meeting (MT)**. Permite realizar el seguimiento de las reuniones, incorporar un acta y registrar el tiempo empleado en cada una
- **Bug (BG)**. Se empleará para realizar el seguimiento de la corrección de defectos encontrados.
- **Corrective Action (CA)**. Cualquier acción correctiva que sea preciso realizar será modelada con esta *issue*.
- **Noncomplieance (NC)**. Las no conformidades detectadas dispondrán de una *issue* particular.

5.4.3. Implementación CMMI y SCRUM: Entorno de desarrollo

Una vez visto el entorno software de gestión veremos ahora qué herramientas y qué arquitectura se empleará para que los analistas y programadores lleven a cabo el trabajo de desarrollo de forma ágil adhiriéndose al mismo tiempo a CMMI.

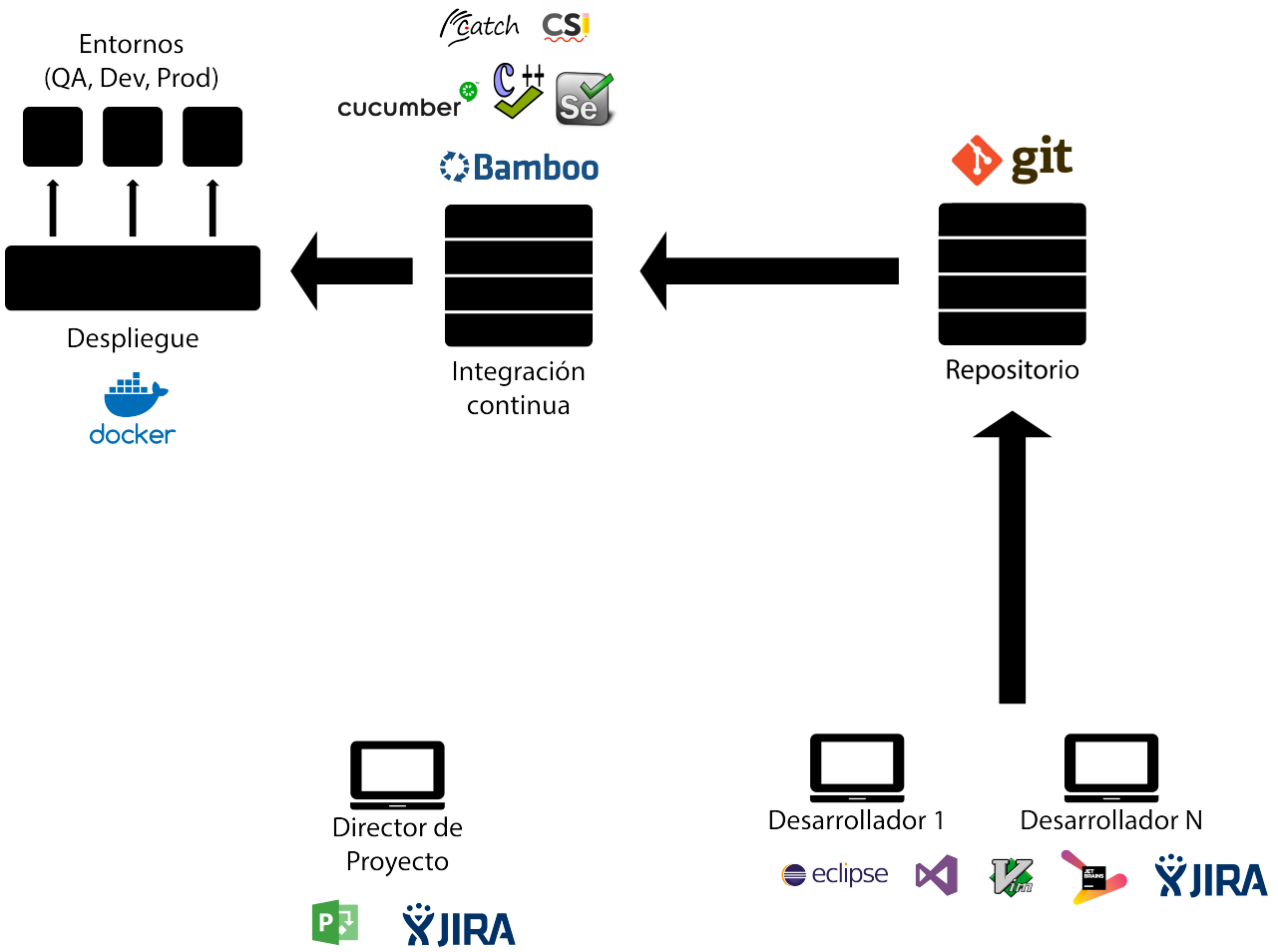

Figura 5.1: Entorno de desarrollo CMMI ágil.

La figura 5.1 resume el entorno de desarrollo propuesto. Veamos a continuación los pasos que se siguen.

1. Los desarrolladores programarán en sus equipos los requisitos contenidos en las historias de usuario mediante el empleo de un *entorno de desarrollo integrado* o IDE. Actualmente los IDE son programas muy potentes que ayudan al programador en la tarea de programar indicándoles errores potenciales mientras escribe código, simplificándole tareas repetitivas, formateando el código de acuerdo a guías de estilo, automatizando procesos como la compilación, la ejecución de tests etc. Es habitual permitir al programador que elige el IDE que quiera usar siempre y cuando el código que se genere sea homogéneo y cumpla con la guía de estilo establecida. La guía de estilo es un documento que debe redactarse antes de comenzar la fase de desarrollo y que estipula qué convención se debe emplear a la hora de formatear el código, el patrón de nombrado de los ficheros, variables, funciones, clases y otros elementos, la codificación de los ficheros, la longitud máxima de las líneas, el espaciado, etc.

2. El código generado es el activo más importante dentro de un proyecto software y por consiguiente debe ser salvaguardado de la mejor manera posible para protegerlo frente a posibles pérdidas intencionadas o no sin que, al mismo tiempo, suponga un lastre o un problema para los desarrolladores. Por esto, se han desarrollado los repositorios de código, que son bases de datos en las que se almacena el código de manera versionada, es decir, cada cambio realizado en un fichero y enviado al repositorio es registrado de forma que es posible recuperar en cualquier momento el fichero en un momento de su historia. Además, los repositorios permiten que un mismo fichero evolucione a partir de un mismo punto de dos o más formas mediante el concepto de ramas y llegado el momento pueda volver a fusionarse. El concepto de ramas se adapta perfectamente a la metodología SCRUM, se puede crear una rama por historia de usuario o por sprint. Existen varias tecnologías y soluciones software, a continuación, se citan las más relevantes y la elegida en nuestro caso:

   a) **Subversion**. Es una tecnología cliente/servidor, esto quiere decir que existe un único almacén (servidor) que guarda el código.

   b) **Git**. Por contraposición a la anterior, esta tecnología es distribuida, esto quiere decir que cada usuario guarda toda la información y los cambios se sincronizan posteriormente. Será nuestra solución elegida ya que se adapta muy bien a las metodologías de desarrollo ágil.

   c) **Mercurial**. Es una solución similar en filosofía a git, aunque con menor cuota de mercado. La descartamos por su bajo nivel de implantación.

   El código fuente estará formado por diferentes lenguajes de programación dependiendo del módulo software. A continuación, enumeramos los lenguajes elegidos por sistema

   a) **Interfaz de usuario frontend**. Existe prácticamente una opción mayoritaria que es el empleo de Javascript junto con alguno de sus librerías más utilizadas que quedará a elección de los programadores.

b) **Interfaz de usuario backend**. Existe una amplia variedad de lenguajes de programación que podrían ser empleados para este módulo, la elección se debe hacer en virtud de los programadores disponibles, la complejidad esperada, el volumen de información a gestionar, la futura escalabilidad y mantenimiento del sistema, etc. Algunos de los lenguajes más habituales son: Java, PHP, Ruby, Python, Scala, Perl y Javascript. En nuestro caso vamos a proponer el empleo del lenguaje Python ya es un lenguaje cada vez más popular, lo que nos garantiza encontrar programadores, permite hacer desarrollos rápidos gracias a su sencillez y al gran número y calidad de las librerías disponibles y presenta un rendimiento aceptable.

c) **Módulo de procesado**. a. El módulo de procesado requiere de un lenguaje eficiente y con capacidad para comunicarse con bases de datos y otros subsistemas. Entre los lenguajes que podríamos emplear tenemos C++, Java o Scala. El primero, C++, es sin duda el más eficiente, pero dispone de pocas librerías disponibles para el procesado de datos. Tanto Java como Scala son dos opciones muy similares y muy empleadas en el procesado de datos. Nos decantaremos por Java por la gran cantidad de programadores disponibles en el mercado.

d) **Fuente de datos**. Los dispositivos encargados de capturar datos son aparatos con serias limitaciones en cuanto a procesador y memoria. Es por esto que un lenguaje como C++ caracterizado por aprovechar al máximo el hardware disponible se convierte en una solución ideal.

3. **Integración continua**. El código almacenado en el repositorio se asemeja a una receta de cocina, contiene la descripción y los pasos para llevar a cabo un plato, pero alguien debe encargarse de realizar ese trabajo. La tarea de traducir el código en software, entendiendo software no solo como un ejecutable capaz de realizar una tarea sino como la suma de ejecutables y documentación asociada, es responsabilidad del módulo de integración continua. Este módulo se encarga de generar la documentación automática, el análisis estático de código (responsable de verificar que las guías de estilo se cumplen y de detectar errores de programación por observación), compilar el código fuente en binarios ejecutables, realizar las pruebas unitarias, funcionales y de integración, generar los componentes entregables y desplegar por último dichos componentes en diferentes entornos y realizar informes con los resultados de todas las operaciones llevadas a cabo. Este módulo cumple con muchos objetivos planteados tanto por las metodologías ágiles como por parte de CMMI. Existen diferentes soluciones software que podrían satisfacer nuestros objetivos, destacamos a continuación, las dos soluciones más populares y utilizadas

   a) **Jenkins**. a. Uno de los más extendidos y populares. Originalmente desarrollador por Sun Microsystems (actual Oracle), permite realizar casi cualquier tarea de integración continua gracias al amplio número de plugins que le permiten interactuar con diferentes herramientas. Podría ser una solución perfectamente válida, es de código abierto y por tanto no se requiere de licencia.

b) **Bamboo**. Es el sistema de integración continua de Atlassian, creador de JIRA. Es similar en prestaciones a Jenkins y al formar parte de la suite de Atlassian nos permite integrarlo con JIRA con mayor facilidad. A diferencia de Jenkins es una herramienta de pago. Será nuestra elección por las facilidades que tendremos para integrarlo con JIRA.

En el módulo de integración continua se llevarán a cabo una serie de pruebas software, tanto pruebas unitarias como funcionales y de integración. Para dar soporte a estas pruebas se emplearán herramientas específicas que dependerán del lenguaje de programación empleado. En el cuadro 5.4 se muestra un resumen de las herramientas por lenguaje a utilizar. Junto con las pruebas el módulo generará documentación de forma automática a partir de los comentarios introducidos por los programadores. La documentación se generar en formato web, de manera que los programadores dispondrán de un enlace web en el que podrán consultar el cometido de cada clase, función o variable documentada. Además, para garantizar la calidad del código se emplearán herramientas que velarán porque el código cumpla las normas definidas en las guías de estilo.

4. **Despliegue**. El módulo de integración continua genera ejecutables, scripts, documentación y otros artefactos que deben ser desplegados para poder ser puestos en marcha. Habitualmente se crea un entorno de desarrollo o pruebas en donde se despliegan los componentes para evaluar su comportamiento o para mostrárselos al cliente antes de llevarlos al entorno real. En cualquier caso, estos entornos requieren de una infraestructura: sistema operativo, servidores web, bases de datos, gestores de colas, etc y la configuración asociada a estos elementos. Para facilitar el despliegue y que este sea automático se empleará software específico. Esto permitirá que las pruebas y entregas de nuevas versiones se automática y muy rápida. Existen diferentes soluciones:

   a) **Docker**. Es un sistema de contenedores que permite desplegar aplicaciones complejas de forma muy sencilla y automática. Los contenedores simulan un entorno de trabajo independientemente del entorno de la máquina. Esto garantiza que la aplicación siempre se despliegue de la misma forma independientemente del sistema sobre el que se ejecute. Será nuestra opción elegida.

   b) **Chef/puppet**. Es un sistema para orquestar el despliegue de una aplicación en base a unas reglas predefinidas.

Lenguaje	**P. Unitarias**	**P. Funcionales**	**Documentación**	**Guía Estilo**
C++	Catch	Google Test	Doxygen	CppCheck
Java	JUnit	Cucumber	Javadocs	Checkstyle
Python	PyTest	Selenium	Sphinx	PEP8
Javascript	Karma	Selenium	JSDoc	JSLint

Cuadro 5.4: Herramientas utilizadas en la integración continua por lenguaje.

5.4.4. Implementación CMMI y SCRUM: Proceso de desarrollo.

Como ya hemos mencionado la gestión del proyecto tendrá tanto una vertiente clásica como una más ágil. La gestión clásica aborda los aspectos de más alto nivel del proyecto, cómo resultado podremos descomponer las actividades a realizar en la estructura de descomposición de tareas conocida como WBS, ver figura 5.2.

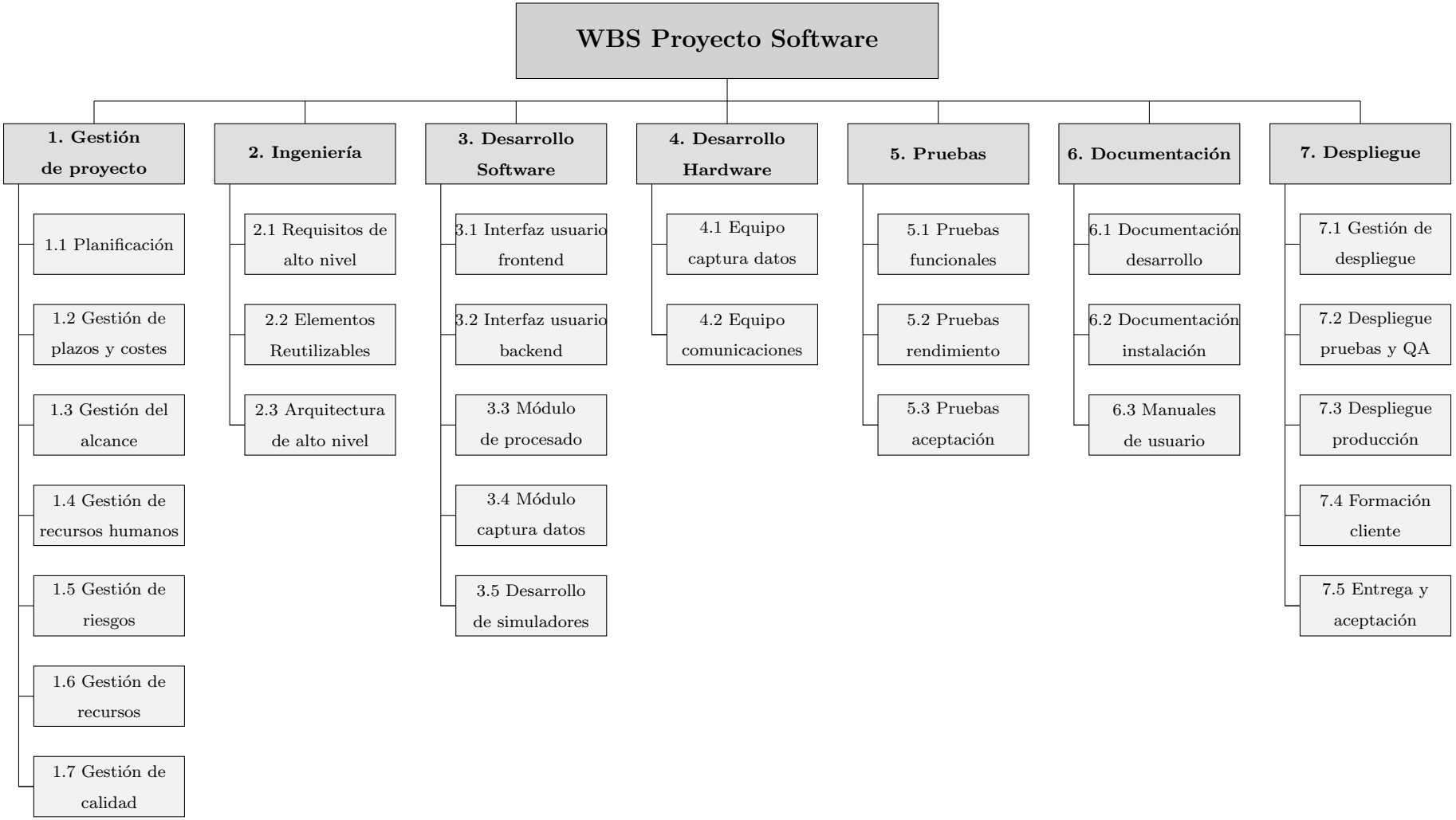

Figura 5.2: Ejemplo WBS de un proyecto software.

El director del proyecto, como responsable del proyecto es quien se encarga de elaborar la WBS, para ello se apoya en Microsoft Project como herramienta de gestión. A partir de este punto, aquellos paquetes de trabajo relacionados con el desarrollo de software, como "Entorno de pruebas", "Interfaz usuario Frontend", "pruebas funcionales", etc se gestionarán a través de JIRA. Mediante un conector JIRA/MS Project el director de proyecto podrá disponer de una actualización de los datos del proyecto en MS Project a partir de los datos introducidos en JIRA.

En este TFM nos centraremos en la parte ágil, por lo que asumiremos que la parte inicial del proyecto ya se ha llevado a cabo. Asumimos por tanto que antes de comenzar el proyecto se ha realizado el estudio de lo que el cliente desea hacer, en virtud de lo cual se hacen estimaciones iniciales de plazos, costes, número de personal requerido y cualificación del mismo. Nos encontramos por tanto en la fase de desarrollo de requisitos y software que hemos decidido hacer de manera ágil mediante SCRUM respetando la filosofía CMMI.

Iteración Inicial

La primera tarea será de la comenzar a profundizar en los requisitos, descomponiendo las ideas y requisitos de alto nivel en otros de más bajo nivel. Para ello se realiza un sprint inicial con el objetivo de definir el "Release plan", es decir, el número de iteraciones, y el número de elementos de alto nivel que se prevén entregar en cada una de las interacciones. Durante el transcurso del sprint aparecerán preguntas y dudas que se consultarán con el cliente, y al finalizar el sprint se realizará una reunión con el propio cliente para que este valide los requisitos recopilados, si bien es cierto que este tipo de reuniones se seguirán realizando a lo largo del proyecto, esta es de vital importancia porque los errores en las etapas iniciales del proyecto tienen un mayor impacto. Esto permite cubrir los objetivos específicos SP1.1 y SP 1.2 del área de proceso Requirements Management: "Comprender los requisitos" y "obtener compromisos para los requisitos". Las reuniones constantes con el cliente en este sprint y sucesivos permite satisfacer el punto SP1.5 "asegurar el alineamiento entre el trabajo y los requisitos".

Iteración de desarrollo

Nos encontramos ahora en pleno proceso de desarrollo y acabamos de comenzar un nuevo sprint SCRUM. Fruto de este sprint elaboraremos una historia de usuario, la cual tendrá una issue JIRA asociada en la que estimaremos en su creación el tiempo en horas que esperamos emplear. Las historias de usuarios serán codificadas para facilitar referirse a ellas.

Una vez definida la historia de usuario el desarrollador creará una subtarea asociada. Esta subtarea es un trabajo concreto y bien definido necesario para satisfacer parte de la historia de usuario, por ejemplo, una subtarea podría consistir en programar la lógica encargada de procesar un formulario web. JIRA generará un código asociado a cada subtarea en el momento de su creación.

El siguiente paso es que el desarrollador programe la lógica y las pruebas unitarias asociadas. Una vez finalice deberá enviar el nuevo código fuente al repositorio

de código, lo que se suele denominar commitear. Todo commit al repositorio va acompañado de un mensaje. El repositorio (git en nuestro caso) verificará que el comentario se ajusta a un patrón específico: "[CODIGOSUBTASK][REQ-XXXX] texto", de esta forma JIRA es capaz de relacionar el código fuente que implementa una subtask de una historia de usuario concreta. Así es como satisfacemos el SP1.4 "mantener la trazabilidad bidireccional de los requisitos" del área de proceso REQM. Por último, los cambios de requisitos se verán registrados en el propio registro de la issue, satisfaciendo el SP 1.3 de REQM.

Corrección de errores y no conformidades

Aunque los errores y no conformidades son conceptos diferentes, ambos son tratados de la misma forma (cada uno a través de su respectiva issue) en JIRA. Los dos son programados y planificados en la reunión del sprint. Los cambios necesarios en el código son commiteados siguiendo el mismo patrón, sin embargo, estos cambios se realizarán en una rama del código diferente y solo una vez el equipo de pruebas da por válida la solución los cambios serán llevados a la rama de desarrollo. Con este mecanismo quedan cubierto el SP 2.2 de PPQA "establecer registros".

Integración continua

Cada noche el servidor Bamboo accederá al repositorio y se descargará la última versión del código fuente. A continuación, procederá compilarlo, y luego a realizar las pruebas. Estas pruebas serán las pruebas unitarias desarrolladas por los programadores y las pruebas funcionales, de integración y rendimiento diseñadas tanto por los programadores como por parte del equipo de QA. Con esto se satisface parcialmente la SP1.2 de PPQA "Evaluación objetiva de productos de trabajo". También se aplican las herramientas responsables de garantizar que el código sigue las guías de estilo y se genera la documentación del código. Como resultado Bamboo generará un informe con los resultados indicando si algún paso no fue satisfactorio. Este informe llegará vía email cada mañana a los responsables del proyecto, de forma que pueda ser discutido en la reunión diaria SCRUM y en caso de existir algún problema se adopten las medidas adecuadas. Las medidas a adoptar se verán reflejadas en algún tipo de issue. Con esto podremos satisfacer parcialmente las SP 2.1 y SP 2.2 del área de proceso PPQA: "Comunicar y resolver no conformidades" y "Establecer registros".

Despliegue

Si las pruebas son superadas satisfactoriamente Bamboo procederá a desplegar la aplicación. Antes de realizar el despliegue Bamboo crear los paquetes e imágenes Docker necesarias y los almacena en un "servidor de versiones". La idea es tener tantas versiones del sistema como compilaciones satisfactorias hemos tenido, con el objetivo de poder realizar análisis de la evolución del sistema. Esto puede ser muy útil a la hora de detectar problemas de rendimiento entre versiones, difíciles de detectar en la etapa de pruebas y que solo aparecen en entornos de pruebas más

complejos o en producción. Estas versiones serán internas al equipo de desarrollo, el cliente recibirá las versiones fruto de un sprint completo, las cuales seguirán un esquema de numerado predefinido.

Capítulo 6

Conclusiones Finales y Líneas de Futuro

6.1. Conclusiones

En este TFM hemos abordado dos temas de gran importante dentro de la ingeniería de software. Por un lado, hemos visto en qué consisten las metodologías ágiles de desarrollo y qué las hace tan diferentes de los métodos de desarrollo tradicionales. Por otra parte, hemos puesto nuestra atención en el proceso software y hemos visto cómo el marco de trabajo CMMI podría ser una herramienta útil para ayudarnos a conseguir que los proyectos software que lleva a cabo una organización ganen en capacidad y madurez. Si nos hubiéramos deteniendo en este punto, el TFM pasaría por ser un mero trabajo descriptivo del estado del arte de las metodologías ágiles y CMMI. Sin embargo, hemos visto cómo es posible que al mismo tiempo que el proceso software se adhiere a la filosofía formal de CMMI, los trabajos de desarrollo puedan llevarse a cabo mediante el empleo de filosofías ágiles.

- Los proyectos software, a diferencia de otros llevados a cabo en otras ramas de la ingeniería, poseen una complejidad inherente
- La ingeniería de software es una rama de la ingeniería que se enfrente a proyectos de enorme complejidad, y por tanto no existen soluciones estandarizadas, formas de trabajo o buenas prácticas que puedan ser aplicables independientemente de la naturaleza del proyecto, la organización o el entorno.
- A la hora de escoger las herramientas más adecuadas para llevar a cabo un proyecto software es importante conocer la dimensión y requisitos del proyecto. Como hemos señalado a lo largo del TFM, los proyectos de software crítico en tiempo real no pueden ser gestionado de igual forma que un proyecto de un portal web.
- La gestión del proceso software es de capital importancia para aumentar el éxito del proyecto. Y por éxito debemos entender que el proyecto cumple

con las expectativas del cliente. Sin embargo, a diferencia de otros tipos de proyectos el software es un producto intelectual por lo que disponer de un personal lo suficientemente capacitado contribuirá en gran medida a alcanzar el éxito.

- CMMI y las metodologías ágiles si se utilizan correctamente pueden ayudar a las organizaciones a que sus proyectos alcancen un equilibrio óptimo. Por un lado, los procesos son controlados, gestionados, medidos y analizados de forma sistemática e institucionalizada, y por otro, la agilidad permite reducir trabas burocráticas, formalismos innecesarios y otros impedimentos que lastran el rendimiento del proyecto.

- Combinar CMMI y agile, puede ayudar también a que las organizaciones sean capaces de responder más rápidamente a cambios en su entorno. Los grandes proyectos software, aquellos que se desarrollan a lo largo de años o que se basan en la evolución de sistemas legados, se pueden beneficiar de la aplicación de metodologías ágiles, ya que están suelen generar sistemas más modulares y fáciles de mantener y evolucionar.

- Cualquier desarrollo profesional de software involucra un número considerable de procesos a desarrollar y por tanto disponer de un marco probado de trabajo como CMMI para optimizar estos procesos es una herramienta que cualquier organización dedicada al desarrollo software debería plantearse usar.

6.2. Líneas de Futuro

Si existe una rama de la ingeniería en constante evolución, sin duda, es la ingeniería de software. Cada año surgen nuevas ideas y tecnologías que desfasan a otras, y en ocasiones, otras abandonadas durante décadas vuelven a tener una segunda oportunidad, en este trabajo hemos mencionado como los métodos incrementales, precursores de los ágiles, habían sido propuesto décadas antes de alcanzar la popularidad y extensión que han tenido en los últimos años. No cabe duda que tanto los métodos ágiles como CMMI evolucionarán en los próximos años y que durante el camino aparecerán nuevas oportunidades para mejorar su sintonía.

En este TFM hemos visto cómo abordar la mejora de procesos con CMMI en combinación con la metodología de desarrollo ágil SCRUM en las áreas de proceso CMMI más importantes correspondiente a los niveles de madurez 2 y 3. Quedaría por explorar cómo pueden ser usados junto a CMMI otras metodologías ágiles como ASD, AUP, XP o DSDM.

Otra posible línea de investigación es cómo afecta a la agilidad del proyecto la adopción de diferentes niveles de madurez. Los niveles 4 y 5 requieren de un control estadístico de los procesos y el uso de otras herramientas cuantitativas que sin duda afectarán de algún modo a la agilidad.

Por último, mencionar que el número de experiencias de implantación de CMMI ágil publicadas es todavía escaso. Es cierto que en los últimos años su número ha

crecido, pero se echa en faltan ciertos aspectos, como cuantificar el impacto que ha tenido la adopción.

Bibliografía

[1] Andrew Hunt. *The pragmatic programmer*. Pearson Education India, 2000.

[2] Project Management Institute. *A Guide to the Project Management Body of Knowledge: PMBOK(R) Guide*. 5th. Project Management Institute, 2013. ISBN: 1935589679, 9781935589679.

[3] EN Iso. «9000: 2005». En: *Quality management systems-Fundamentals and vocabulary (ISO 9000: 2005)* (2005), pág. 1.

[4] CMMI Product Team. *CMMI for Development, Version 1.3 (CMU/SEI-2010-TR-033). Software Engineering Institute*. 2010.

[5] International Project Management Association, Gilles Caupin y col. *IPMA competence baseline: ICB; Version 3.0*. Internat. Project Management Association, 2006.

[6] AN ISO. «21500: 2012». En: *Guidance on project management* (2012).

[7] Edsger W Dijkstra. «The humble programmer». En: *Communications of the ACM* 15.10 (1972), págs. 859-866.

[8] Pierre Bourque, Richard E Fairley y col. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

[9] James Johnson y col. «The chaos report». En: *Standishgroup. com* (1994).

[10] Magne Jørgensen y Kjetil Moløkken-Østvold. «How large are software cost overruns? A review of the 1994 CHAOS report». En: *Information and Software Technology* 48.4 (2006), págs. 297-301.

[11] Kjetil Molokken y Magen Jorgensen. «A review of software surveys on software effort estimation». En: *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on*. IEEE. 2003, págs. 223-230.

[12] J Laurenz Eveleens y Chris Verhoef. «The rise and fall of the chaos report figures». En: *IEEE software* 27.1 (2010), págs. 30-36.

[13] Barry W. Boehm. «Software Engineering-as It is». En: *Proceedings of the 4th International Conference on Software Engineering*. ICSE '79. Munich, Germany: IEEE Press, 1979, págs. 11-21. URL: `http://dl.acm.org/citation.cfm?id=800091.802916`.

[14] Ian Sommerville. «Software Engineering. International computer science series». En: *ed: Addison Wesley* (2004).

[15] Richard H Thayer, Sidney C Bailin y M Dorfman. *Software requirements engineerings*. IEEE Computer Society Press, 1997.

[16] Barry Boehm. «Software risk management». En: *ESEC'89* (1989), págs. 1-19.

[17] Vincent Massol y Ted Husted. *JUnit in action*. Manning Publications Co., 2003.

[18] Winston W Royce y col. «Managing the development of large software systems». En: *proceedings of IEEE WESCON*. Vol. 26. 8. Los Angeles. 1970, págs. 1-9.

[19] Herbert D Benington. «Production of large computer programs». En: *Annals of the History of Computing* 5.4 (1983), págs. 350-361.

[20] Thomas E Bell y Thomas A Thayer. «Software requirements: Are they really a problem?» En: *Proceedings of the 2nd international conference on Software engineering*. IEEE Computer Society Press. 1976, págs. 61-68.

[21] Gene Kim. «DevOps distilled, Part 1: The three underlying principles». En: (2013). URL: `https://www.ibm.com/developerworks/library/se-devops/part1/part1-pdf.pdf`.

[22] Barry Boehm. «A spiral model of software development and enhancement». En: *ACM SIGSOFT Software Engineering Notes* 11.4 (1986), págs. 14-24.

[23] Barry W. Boehm. «A spiral model of software development and enhancement». En: *Computer* 21.5 (1988), págs. 61-72.

[24] Per Kroll y Philippe Kruchten. *The rational unified process made easy: a practitioner's guide to the RUP*. Addison-Wesley Professional, 2003.

[25] UML Revision Taskforce. «OMG UML Specification v. 1.4». En: *Object Management Group* (2001).

[26] Ivar Jacobson y col. *The unified software development process*. Vol. 1. Addison-wesley Reading, 1999.

[27] Scott Ambler, John Nalbone y Michael Vizdos. *Enterprise unified process, the: extending the rational unified process*. Prentice Hall Press, 2005.

[28] Harlan D Mills, Michael Dyer y Richard C Linger. «Cleanroom software engineering». En: *IEEE Software* 4.5 (1987), pág. 19.

[29] Jane Radatz, Myrna Olson y Stuart Campbell. «Mil-std-498». En: *Crosstalk, the Journal of Defense Software Engineering* 8.2 (1995), págs. 2-5.

[30] Leanna Rierson. *Developing safety-critical software: a practical guide for aviation software and DO-178C compliance*. CRC Press, 2013.

[31] Craig Larman y Victor R Basili. «Iterative and incremental developments. a brief history». En: *Computer* 36.6 (2003), págs. 47-56.

[32] Ernest A Edmonds. «A process for the development of software for non-technical users as an adaptive system». En: *General Systems* 19.215C18 (1974), pág. 8.

[33] Justus D Naumann y A Milton Jenkins. «Prototyping: the new paradigm for systems development». En: *Mis Quarterly* (1982), págs. 29-44.

[34] James Martin. *Rapid application development*. Macmillan Publishing Co., Inc., 1991.

[35] K. Beck y col. *The Agile Manifesto*. Inf. téc. The Agile Alliance, 2001.

[36] A Cockburn y col. «The Declaration of Interdependence for Modern Management or DOI». En: *Online: http://alistair. cockburn. us/The+ declaration+ of+ interdependence+ for+ modern+ management+ or+ DOI* (2005).

[37] RC Martin y col. «Manifesto for software craftsmanship». En: *Online: http://manifesto. softwarecraftsmanship. org (10.10. 2015)* (2009).

[38] Ken Schwaber y Jeff Sutherland. «The scrum guide». En: *Scrum Alliance* 21 (2011).

[39] Scott Ambler. *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons, 2002.

[40] Jim Highsmith. *Adaptive software development: a collaborative approach to managing complex systems*. Addison-Wesley, 2013.

[41] Steve R Palmer y Mac Felsing. *A practical guide to feature-driven development*. Pearson Education, 2001.

[42] Kent Beck. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[43] Mary Poppendieck y Tom Poppendieck. *Lean Software Development: An Agile Toolkit: An Agile Toolkit*. Addison-Wesley, 2003.

[44] H Takeuchi e I Nonaka. «16 The new new product development game». En: *Japanese Business: Part 1, Classics Part 2, Japanese management Vol. 2: Part 1, Manufacturing and production Part 2, Automotive industry Vol. 3: Part 1, Banking and finance Part 2, Corporate strategy and inter-organizational relationships Vol. 4: Part 1, Japanese management overseas Part 2, Innovation and learning* 64.1 (1998), pág. 321.

[45] Ken Schwaber. «Scrum development process». En: *Business Object Design and Implementation*. Springer, 1997, págs. 117-134.

[46] Ken Schwaber y Mike Beedle. *Agile software development with Scrum*. Vol. 1. Prentice Hall Upper Saddle River, 2002.

[47] David J Anderson. *Kanban: successful evolutionary change for your technology business*. Blue Hole Press, 2010.

[48] David J Anderson y Dragos Dumitriu. «From Worst to Best in 9 Months». En: *TOC ICO World Conference*. 2005.

[49] Corey Ladas. «Scrumban». En: *Lean Software Engineering-Essays on the Continuous Delivery of High Quality Information Systems* (2008).

[50] Mark Paulk. «Capability maturity model for software». En: *Encyclopedia of Software Engineering* (1993).

[51] Mark C Paulk y col. «Capability maturity model, version 1.1». En: *IEEE software* 10.4 (1993), págs. 18-27.

[52] Bill Curtis, Bill Hefley y Sally Miller. *People capability maturity model (P-CMM) version 2.0*. Inf. téc. DTIC Document, 2009.

[53] Roger Bate y col. *A systems engineering capability maturity model, Version 1.1*. Inf. téc. DTIC Document, 1995.

[54] Alec Dorling. «SPICE: Software process improvement and capability determination». En: *Software Quality Journal* 2.4 (1993), págs. 209-224.

[55] Pasi Kuvaja y Adriana Bicego. «BOOTSTRAP—a European assessment methodology». En: *Software Quality Journal* 3.3 (1994), págs. 117-127.

[56] SEI. *Capability Maturity Model Integration, version 1.1*. Inf. téc. CMU/SEI-2002-TR-011. Pittsburgh, Carnegie Mellon University: Software Engineering Institute, 2002. URL: `http://www.sei.cmu.edu/cmmi/`.

[57] CMMI Product Team. «CMMI for Development, version 1.2». En: (2006).

[58] CMMI Product Team. *CMMI for Services Version 1.3*. Lulu. com, 2011.

[59] CMMI Product Team. *CMMI for Acquisition Version 1.3*. Lulu. com, 2011.

[60] W Edwards Deming. «Out of the crisis, Massachusetts Institute of Technology». En: *Center for advanced engineering study, Cambridge, MA* 510 (1986).

[61] Philip B Crosby. *Quality is free: The art of making quality certain*. Signet, 1980.

[62] Joseph M Juran y col. *Juran on planning for quality*. Free Press New York, 1988.

[63] Victor R Basili y H Dieter Rombach. «The TAME project: Towards improvement-oriented software environments». En: *IEEE Transactions on software engineering* 14.6 (1988), págs. 758-773.

[64] Victor R Basili. *Software modeling and measurement: the Goal/Question/Metric paradigm*. Inf. téc. 1992.

[65] Kevin Pulford, Annie Kuntzmann-Combelles y Stephen Shirlaw. *A quantitative approach to software management: the AMI handbook*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[66] Stephen Viller y Ian Sommerville. «Ethnographically informed analysis for software engineers». En: *International Journal of Human-Computer Studies* 53.1 (2000), págs. 169-196.

[67] David Martin y col. «'Good'organisational reasons for'Bad'software testing: An ethnographic study of testing in a small software company». En: *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society. 2007, págs. 602-611.

[68] John Hughes y col. «Moving out from the control room: ethnography in system design». En: *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM. 1994, págs. 429-439.

[69] Business Process Model. «Notation (BPMN) version 2.0». En: *OMG Specification, Object Management Group* (2011).

[70] Ian Sommerville y Pete Sawyer. *Requirements engineering: a good practice guide*. John Wiley & Sons, Inc., 1997.

[71] Hillel Glazer y col. «CMMI or agile: why not embrace both!» En: (2008).

[72] David J Anderson. «Stretching agile to fit CMMI level 3-the story of creating MSF for CMMI/spl reg/process improvement at Microsoft corporation». En: *Agile Conference, 2005. Proceedings*. IEEE. 2005, págs. 193-201.

[73] Michael A Cusumano y Richard W Selby. *Microsoft secrets*. 1997.

[74] Joseph P Elm y col. «A Survey of Systems Engineering Effectiveness-Initial Results». En: (2008).

[75] Jessica Diaz, Juan Garbajosa y Jose A Calvo-Manzano. «Mapping CMMI level 2 to scrum practices: An experience report». En: *European Conference on Software Process Improvement*. Springer. 2009, págs. 93-104.

[76] CJ Salinas, Maria J Escalona y Manuel Mejias. «A scrum-based approach to CMMI maturity level 2 in web development environments». En: *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services*. ACM. 2012, págs. 282-285.

[77] K Łukasiewicz y J Miler. «Improving agility and discipline of software development with the Scrum and CMMI». En: *IET software* 6.5 (2012), págs. 416-422.

[78] Taghi Javdani Gandomani y col. «Compatibility of agile software development methods and CMMI». En: *Indian Journal of Science and Technology* 6.8 (2013), págs. 5089-5094.

[79] Marco Palomino y col. «Agile Practices Adoption in CMMI Organizations: A Systematic Literature Review». En: *International Conference on Software Process Improvement*. Springer. 2016, págs. 57-67.

[80] Philipp Diebold, Thomas Zehler y Dominik Richter. «How Do Agile Practices Support Automotive SPICE Compliance?» En: (2017).

[81] Luigi Benedicenti y col. «Improved Agile: A Customized Scrum Process for Project Management in Defense and Security». En: *Software Project Management for Distributed Computing*. Springer, 2017, págs. 289-314.

[82] Carlos Joaquin Torrecilla Salinas. «A mature agile approach in web engineering contexts». Tesis doct. Universidad de Sevilla, 2017.

[83] Carlos J Torrecilla-Salinas y col. «An Agile approach to CMMI-DEV levels 4 and 5 in Web development». En: (2016).

[84] Gabriela Arauz Ortiz y col. «Integrating Agile Methods into a Level 5 CMMI-DEV Organization: a Case Study». En: *IEEE Latin America Transactions* 14.3 (2016), págs. 1440-1446.