Gestión de Proyectos de Desarrollo de Software

Bibliografía:

- Universidad de Murcia. Apuntes de Gestión de Procesos de Desarrollo de Software.
- Wikipedia, the Free Encyclopedia. *Standards organization, Technical standard, De facto standard*, *ISO/IEC 15504*, *ReqIF*, *PMBOK*, *Earned value management*, *COCOMO*, *Precedence Diagram Method*, *Arrow Diagramming Method*.
- Diccionario de la Real Academia Española. *De facto*.

Capítulo 1

Ingeniería de software

Según el Vocabulario de la Ingeniería del Software y de Sistemas, ISO/IEC/IEEE 24765:2017, la **ingeniería de software** es «la aplicación sistemática de conocimiento científico y tecnológico, métodos y experiencia al diseño, implementación, pruebas y documentación del software». Es **sistemática**, con planes y procedimientos metódicos; **disciplinada**, sujeta a control respecto a ciertos estándares, y **cuantificable**, con realización y resultados medibles.

1.1. Dimensiones

El SEI clasifica el conocimiento en ingeniería de software en 4 dimensiones mediante 2 visiones:

1. **Visión proceso**, propuesta para estructurar su corpus de conocimiento en base a dos dimensiones:

   a) **Actividad**, que clasifica las actividades realizadas en ingeniería de software:

      1) **Desarrollo:** Actividades que producen artefactos, como análisis de requisitos, especificación, diseño, implementación y pruebas.
      2) **Control:** Actividades que moderan el desarrollo, como evolución del software (control de versiones, cambios, gestión de configuraciones, etc.) y calidad del software (aseguramiento, control y prueba, etc.).
      3) **Gestión:** Actividades que implican a ejecutivos, administrativos y supervisores, incluyendo las actividades técnicas de soporte a su trabajo, como planificación de proyectos, estimación de costes, asignación de recursos, constitución de equipos o asuntos legales y de contratación.
      4) **Operaciones:** Actividades relacionadas con el uso del software en la organización, como la formación, la planificación de la puesta en marcha, la puesta en marcha, la transición al nuevo sistema, la operación del software y la gestión de su retirada.

   b) **Aspecto**, con los aspectos similares de cada actividad:

1) **Abstracciones:** Principios fundamentales como ocultación de información o principios de diseño, y modelos formales como de proceso de desarrollo (en cascada, prototipado, etc.), de computación secuencial y concurrente (máquinas de estado finito, redes de Petri, etc.) o de estimación de costes (COCOMO, etc.)
2) **Representación:** Notaciones como tablas de decisión, DFD o PERT, y lenguajes como Ada.
3) **Métodos:** Métodos formales como pruebas de corrección para la verificación o especificaciones ejecutables; intuitivos como AE, AOO o DOO, y prácticas comunes como la programación estructurada para la implementación).
4) **Herramientas:** Integradas o de software individual, como correo electrónico, procesador de textos, CASE o compiladores.
5) **Evaluación:** Métricas, análisis y evaluación de los productos y el proceso software y del impacto en las organizaciones, para mejorar futuros desarrollos.
6) **Comunicación:** Oral y escrita en forma de documentación o formación.

2. **Visión producto**, que amplía la presentación unidimensional basada en el ciclo de vida típico de análisis de requisitos, especificación, diseño, implementación, prueba y mantenimiento con dos dimensiones:

   a) **Clase de sistema software**, con características específicas de un software concreto, por ejemplo el software concurrente, para el que existen distintos lenguajes y notaciones de diseño y especificación. Los grupos resultantes de la clasificación no tienen por qué ser disjuntos.

   Según el dominio distinguimos sistemas de comunicación, de aviación, sistemas operativos, bases de datos, etc. Según el tipo de relación con el entorno distinguimos sistemas por lotes, reactivos, interactivos, de tiempo real, embebidos, etc.

   b) **Requisitos del sistema**, con las propiedades que debería tener el sistema a construir. Se suele restringir a aspectos funcionales, pero la consecución de los no funcionales depende de muchas de las actividades realizadas en el proceso.

1.2. Informes CHAOS

Son informes de pago de *Standish Group*, liberados cuando tienen cierta antigüedad, sobre el desarrollo de la ingeniería de software, y clasifican los proyectos en:

- **Éxitos:** Se entregan a tiempo dentro de su presupuesto y con las funciones requeridas.
- **Deficientes:** Se entregan tarde, sobrepasando el coste o sin todas las funciones requeridas.
- **Fracasos:** Se cancelan antes de terminar o se entregan pero nunca se usan.

Según el informe de 2020, el 31 % de los proyectos son éxitos, el 50 % son deficientes y el 19 % son fracasos.

1.3. Procesos software

Un **artefacto** es algo tangible creado con un propósito práctico. Una **actividad** es una actuación de un agente con ciertos objetivos.

Un **proceso software** es un conjunto coherente de estructuras organizativas, políticas, prácticas, tecnologías, artefactos y transformaciones usados para concebir, desarrollar, implantar y mantener software y productos asociados, como planes de proyecto, documentos de requisitos, documentos de análisis o diseño, codificación, casos de prueba, manuales de usuario, etc. Está formado por:

1. Un **proceso de desarrollo**, con las actividades necesarias para crear un sistema software que satisfaga ciertas necesidades de los usuarios.
2. Una **gestión del proyecto** para planificar y controlar las actividades de desarrollo.
3. Un **control de configuración**, que gestiona los cambios para mantener los objetivos de calidad, costo e integridad de los productos.

El **ciclo de vida** de un producto o proyecto software es su evolución desde su concepción hasta que deja de usarse, y puede describirse según las actividades que se realizan en él.

El **ciclo de desarrollo** de un producto software es la parte de su ciclo de vida desde el análisis hasta la entrega.

1.4. Estrategias de desarrollo de software

Un **método** es la especificación de una secuencia de acciones orientadas a un cierto propósito, y determina el orden y la forma de llevar a cabo unas actividades. Una **metodología** es un conjunto coherente de métodos relacionados por principios comunes. Algunas metodologías:

1. **En cascada:** se hacen las fases en orden secuencial. Útil cuando los requisitos son fijos y el trabajo avanza de forma lineal, al contrario que en la actualidad.
2. **Iterativo** e **incremental:** con iteraciones en las que se hacen todas las fases para lo siguiente a implementar.
3. **MDD** (***Model-Driven Development***): La principal salida del proceso son los modelos, no los programas. Un **PSM** (***Platform-Specific Model***) describe una solución desde la perspectiva de una plataforma concreta. También hay **CIMs** (*Computing-Independent Models*) y **PIMs** (*Platform-Independent Models*).

Según una encuesta de Ambler, las metodologías más exitosas son *lean*, iterativa y ágil.

1.5. Modelos de mejora

Un **modelo de proceso** es una colección estructurada de prácticas que describen características de procesos que han probado por experiencia ser efectivas. Los procesos usados dependen del dominio de aplicación, la estructura y tamaño de la organización, etc., y generalmente se dividen en tradicionales y ágiles.

Un **modelo de mejora de procesos** o **modelo de madurez y capacidad** sirve para estudiar los procesos usados en una organización y basarse en esto para mejorarlos, pasando de procesos ad-hoc inmaduros a procesos disciplinados y maduros con mejor calidad y eficacia, mejorando la productividad y la calidad, alargando el ciclo de vida y logrando planificaciones y presupuestos más precisos y previsibles.

Algunos modelos de mejora son:

- **CMM** (*Capability Maturity Model*), de CMU (*Carnegie-Mellon University*) y SEI (*Software Engineering Institute*), un marco de evaluación y guía en la mejora de proceso de desarrollo de software, suplantado por **CMMi**.
- **ISO 9000**, un sistema de gestión de calidad.
- **PSP** (*Personal Software Process*).
- **TSP** (*Team Software Process*).

1.6. Estandarización

Un **estándar oficial** es un requisito técnico aprobado por un organismo de estandarización, como puede ser **ACM** (*Association for Computer Machinery*), **IEEE** (*Institute of Electrical and Electronic Engineers*), **ISO** (*International Standardization Organization*) o **IEC** (*International Electrotechnical Commission*).

Un **estándar *de facto*** es una convención no aprobada por un organismo de estandarización pero ampliamente usada, como LaTeX o el formato de Microsoft Word 2003.

Una **guía** es un conjunto de criterios bien definidos y documentados que encaminan una actividad, siendo más flexible que un estándar.

Una organización de software puede adoptar modelos de mejora de procesos estándar para obtener certificados de calidad y mejorar su imagen pública, por exigencias de calidad de los clientes y el estado y como resultado de la cultura impuesta por las propuestas de evaluación y mejora en la industria.

ISO/IEC 33000 es una familia de normas de evaluación de procesos en ingeniería en general, que suplanta al estándar **ISO/IEC TR 15504** en el que se basaba el modelo de mejora **SPICE** (*Software Process Improvement and Capability dEtermination*). No indica los modelos concretos a usar para evaluar la madurez en el desarrollo de software.

La versión 2.0 de **MMIS** (Modelo de Madurez de Ingeniería de Software), de AENOR, es compatible con la primera y amplía el alcance basándose en el estándar ISO/IEC/IEEE 12207:2017 *"Systems and software engineering—Software life cycle processes"* para adaptar ISO/IEC 33000 a los procesos de software.

ITMark es una certificación orientada a PYMES del grupo TECNALIA del ESI, con presencia en más de 16 países en Europa y América latina, que combina modelos y estándares de calidad y mejora de procesos como CMMi, ISO 20000, **ISO 27000** de seguridad y **EFQM** (*European Foundation for Quality Management*).

Capítulo 2

CMMi

CMMi es un modelo de mejora estándar que puede usarse para mejorar los procesos de desarrollo y mantenimiento de software o determinar el nivel de madurez de una organización de software. Integra y suplanta a **CMM-SW** (*CMM for Software*), **SE-CMM** (*Systems Engineering Capability Maturity Model*), **IPD-CMM** (*Integrated Product Development*, que intenta integrar los dos anteriores eliminando duplicación) y, a partir de CMMi 1.1, **CMMi-SS** (*Supplier Sourcing: Outsourcing*). Busca averiguar qué hacer para mejorar la productividad y alinear las operaciones con los objetivos de negocio, aunque no dice cómo implementar las prácticas.

CMM y CMMi estaban disponibles gratuitamente hasta CMMi 2.0, que requiere pagar una licencia de compra. Ambos fueron patrocinados por el Departamento de Defensa de Estados Unidos, aunque este ya no financia las actualizaciones de CMMi ni respalda su uso. Además, CMM estuvo disponible antes que ISO/IEC 15504 y CMMi incorporó muchas ideas de este, por lo que CMMi tuvo más éxito que ISO/IEC 15504.

CMMi puede aplicarse a organizaciones de cualquier tamaño, habiéndose aplicando a empresas con menos de 25 empleados y con más de 1000, siendo lo más común de 2008 a 2019 las empresas de entre 26 y 100 empleados. Se puede aplicar a todo tipo de actividad y sector, siendo los más comunes finanzas, transporte, servicios públicos y telecomunicaciones.

Los países con más certificaciones CMMi, en orden decreciente, son China, Estados Unidos e India, cada vez con más certificaciones. España está en quinto lugar y es el país con más certificaciones de Europa.

2.1. Estructura

CMMi 2.0 está formado por 5 **componentes**:

- **Modelo**, formado por un modelo básico, contexto específico y vínculos a material externo. Incluye **vistas**, selecciones del modelo que permiten centrarse en lo que se considera importante para la empresa, de las que DEV, SVC y **SPM** o **SM** (*Supplier Management*) sustituyen, respectivamente, a las áreas de interés DEV, SVC y ACQ de CMMi 1.3, lo que ayuda a reducir la redundancia. Las empresas pueden construir vistas personalizadas si lo desean.

- **Método de evaluación**, renovado para aumentar la confiabilidad y reducir el coste.
- **Capacitación y certificación**, dirigida a objetivos de aprendizaje, con componentes modulares y opciones en persona y virtuales.
- **Sistemas y herramientas**, con un inicio de sesión único para acceder a los modelos en línea y los recursos.
- **Guía de adopción**, incluyendo la transición de CMMi 1.3 a CMMi 2.0.

2.2. Definiciones

Un **producto de trabajo** es un artefacto: un resultado de un proceso, actividad o tarea, independiente o parte de una solución.

Un **producto** es un producto de trabajo que está previsto entregar a un cliente o usuario final, y un **servicio** (*service*) es un producto intangible y no almacenable.

2.3. Modelo

El modelo consta de 25 **áreas de práctica**, colecciones de prácticas similares que logran una intención y obtienen cierta información, de las que 20 aparecen en la vista DEV.

Un **grupo de práctica** es una colección de prácticas de una misma área de práctica para ayudar a su comprensión y adopción. Cada grupo de práctica tiene una serie de **niveles de evolución**, de 0 a un máximo no superior a 5 según el área de práctica. Los niveles son los siguientes, y cada uno amplía el anterior con nuevos requisitos de funcionalidad y sofisticación.

0. **Incompleto.** Se cumple siempre.
1. **Inicial.**
2. **Administrado.**
3. **Definido.**
4. **Administrado cuantitativamente.**
5. **Optimizado.**

Las áreas de práctica se agrupan en **áreas de capacidad**, con áreas de práctica relacionadas que pueden proporcionar un mejor rendimiento en las habilidades y actividades de los proyectos de la organización, y que a su vez se agrupan en 4 **categorías**, vistas que agrupan áreas de capacidad y otras vistas para abordar un problema común detectado en el desarrollo de entregables por parte de empresas.

A continuación vemos las categorías con algunas de sus áreas de capacidad y algunas de las áreas de práctica de estas, a veces con el nivel de evolución máximo en el área.

2.3.1. Hacer

Producir y entregar soluciones de calidad.

1. **Garantizar la calidad** (**ENQ**, *ENsuring Quality*).
   a) **Administración y desarrollo de los requisitos** (**RDM**, *Requirements Development & Management*, máx. 3).
   b) **Verificación y validación** (**VV**, *Verification & Validation*, máx. 3).

2.3.2. Gestionar

Planificar y gestionar la implementación de soluciones.

1. **Administrar la resiliencia del negocio** (**MBR**, *Managing Business Resilience*).
   a) **Gestión de riesgos y oportunidades** (**RSK**, *Risk & Opportunity Management*, máx. 3).
2. **Planificación y administración de trabajos** (**PMW**, *Planning & Managing Work*)
   a) **Estimación** (**EST**, *ESTimating*, máx. 3).
   b) **Monitoreo y control** (**MC**, *Monitor & Control*, máx. 3).
   c) **Planificación** (**PLAN**, *PLANning*, máx. 4).

2.3.3. Habilitar

Apoyar la implementación y la entrega.

2.3.4. Mejorar

Mantener y mejorar el rendimiento de la empresa.

1. **Mantener el hábito y la persistencia** (**SHP**, *Sustaining Habit and Persistence*).
   a) **Gobernanza** (**GOV**, *GOVernance*, máx. 4). Orientar a los administradores en el patrocinio y la gobernanza de las actividades de proceso.
   b) **Infraestructura de implementación** (**II**, *Implementation Infrastructure*, máx. 3). Asegurarse de que los procesos importantes para loa organización son usados y mejorados de forma habitual.

2.4. Niveles de certificación

CMM y CMMi describen niveles de mejora desde procesos inmaduros a procesos disciplinados y maduros con mejor calidad y eficiencia.

Nivel	CAR	DAR	RSK	OT	PCM	PAD	PR	VV	TS	PI	MPM	PQA	CM	MC	PLAN	EST	RDM	GOV	II
2											2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3		3	3	3	3	3	3
4	4				4						4				4			4	
5	5										5								

Cuadro 2.1: Requisitos para los niveles 2–5 en DEV sin SAM.

2.4.1. Niveles de capacidad

Nivel de sofisticación de los procesos que marca el **grado de institucionalización** de los procesos en la organización, según la **persistencia**, la obstinación en la práctica pese a las dificultades encontradas, y el **hábito**, la dificultad de abandonar las prácticas. Los niveles van de 0 a 3:

0. **Incompleto.** Rendimiento inconsistente.
1. **Inicial.** Aborda problemas de rendimiento.
2. **Gestionado.** Identifica y monitoriza el progreso hacia los objetivos de rendimiento del proyecto.
3. **Definido.** Se centra en lograr los objetivos de rendimiento del proyecto y la organización.

2.4.2. Niveles de madurez

Niveles de 0 a 5 que comprenden un conjunto predefinido de áreas de práctica. Las organizaciones en los niveles 0 y 1 dirigen los proyectos informalmente y con logros por aportaciones puntuales, y los superiores cuentan con personas creativas y bien formadas y procesos de desarrollo y gestión bien definidos para alcanzar logros de forma sistemática.

Para la vista DEV sin SAM, los requisitos de los niveles superiores aparecen en el cuadro 2.1.

2.5. Evaluaciones

Hay 4 tipos:

1. ***Benchmark Appraisal*** (**SCAMPI A** en CMMi 1.3): Permite identificar oportunidades para mejorar en la adopción de los procesos y el rendimiento empresarial, y es válida por 3 años.
2. ***Sustainment Appraisal***: Se hace después después del *Benchmark Appraisal* para validar las evaluaciones pasadas, y vale por 2 años.
3. ***Action Plan Reappraisal***: Segunda oportunidad después de no alcanzar el nivel esperado en un *Benchmark Appraisal* o un *Sustainment Appraisal*.

4. ***Evaluation Appraisal*** (**SCAMPI B** o **C** en CMMi 1.3): Evaluación informal y flexible para ayudar a las empresas a prepararse para el *Benchmark Appraisal*.

2.5.1. Resultados

El resultado de una evaluación en CMMi es una puntuación de la organización, que puede ser:

- Un **perfil de nivel de capacidad** (*capability level profile*), una **representación continua** del grado de disciplina de proyectos formada por una lista de áreas de práctica y el nivel de capacidad alcanzado en cada una. Es útil para mejorar problemas concretos en la organización más cercanos a sus objetivos.

  Un **perfil alcanzado** (*achievement profile*) representa el progreso real de la organización en las áreas de práctica consideradas, y un **perfil objetivo** (*target profile*) representa los objetivos de mejora planificados.

- Un **nivel de madurez** (*maturity level*), una **representación escalonada**, más guiada, útil cuando no hay un área de práctica de preferencia pero se busca la mejora gradual y sistemática. Para conseguir un nivel hay que tener previamente el anterior, salvo que se puede pasar directamente del 3 al 5.

  La mayoría de certificaciones de este tipo de 2008 a 2019 son de nivel 3.

Las dos representaciones pueden coexistir en una organización, a la vez o en momentos distintos, pues el trabajo hecho con una se puede usar para la otra.

2.6. Formato

La explicación de un área de práctica se organiza en:

1. Nombre del área e icono.
2. ***Required PA Information***
   a) ***Intent*:** los resultados que se esperan.
   b) ***Value*:** valor comercial alcanzable con la adopción de las prácticas.
   c) ***Additional Required PA Information*:** opcional, para comprender mejor el área.
3. ***Explanatory PA Information***
   a) ***Practice Summary*,** con un resumen de los grupos de práctica y lo que se requiere de cada uno en cada nivel.
   b) ***Additional PA Explanatory Information***
   c) ***Related Practice Areas*,** si es necesario. Algunas áreas relacionadas.
   d) ***Context Specific*,** si aplica.

4. Grupos de prácticas. Estructuran las prácticas según una función lógica, un tema o un hilo, para facilitar su comprensión y adopción.

Una práctica se organiza en:

1. ***Required Practice Information***
   a) ***Practice Statement***
   b) ***Value*:** valor comercial alcanzable con su adopción.
   c) ***Additional Required Information*:** describe con detalle el alcance y la intención de la práctica para facilitar la comprensión y dar una interpretación clara y coherente.
2. ***Explanatory Information***
   a) ***Additional Explanatory Information***
   b) ***Example Activities***
   c) ***Example Work Products***
   d) **Áreas de práctica relacionadas,** si es necesario.
   e) **Información específica de contexto,** si aplica. De cada contexto incluido aparecen una identificación y una descripción, material informativo adicional, ejemplos de actividades, ejemplos de productos de trabajo y áreas de práctica relacionadas.
   f) **Material externo:** Material informativo ampliado con ejemplos de la práctica en dominios específicos, activos de la práctica, plantillas del proceso de implementación, descripciones de procesos, módulos de entrenamiento, herramientas, métodos, casos de estudio, etc.

Capítulo 3

Ingeniería de requisitos

Un **requisito** es una declaración muy específica, precisa y sin ambigüedad de una o más necesidades particulares a satisfacer por un producto, junto a sus limitaciones y condiciones. Según Alan Davis, un requisito define un objeto, función o estado; limita o controla las acciones asociadas a uno, o define relaciones entre objetos, funciones y estados.

Un **accionista** o **parte interesada** es una persona, un grupo o una organización con interés directo o indirecto en un producto o proyecto, como un usuario, cliente, jefe de proyecto, ingeniero de requisitos, analista o programador.

Una **regla de negocio** es una condición que se debe satisfacer cuando se hace una actividad de negocio. El software puede ignorar la regla para que los usuarios la manejen manualmente; avisar al usuario cuando esta se incumple para que el usuario decida si incumplirla o no, o forzar su cumplimiento con algún mecanismo de excepción.

La **ingeniería de requisitos** es la parte de la ingeniería de software que comprende las actividades de desarrollo relacionadas con la gestión y definición de requisitos para sistemas software, mediando los dominios de adquisidor y proveedor. Sus objetivos son:

1. Alcanzar un acuerdo entre clientes, desarrolladores y usuarios sobre lo que hay que desarrollar.
2. Ser la base para el diseño de software.
3. Permitir la verificación y validación de los productos obtenidos.
4. Orientar a potenciales clientes sobre la definición de los productos mediante catálogos de requisitos alternativos.

Se le suele dedicar el 10 % del esfuerzo de un proyecto, aunque dedicar más esfuerzo disminuye el tiempo total del proyecto según estudios como los del informe SERENA.

3.1. Importancia

Según Robert Glass, las principales causas de fracaso de proyectos de software son la inestabilidad de los requisitos, pues en general los clientes o usuarios no tienen realmente claro qué

quieren o qué problema que quieren resolver, y una mala estimación de el tiempo y el coste del proyecto.

Los errores en las primeras fases del desarrollo son más caros de corregir porque pasa más tiempo hasta terminar las fases y obtener retroalimentación, siendo los más caros los errores en los requisitos. Algunos son:

1. No descubrir requisitos relevantes a tiempo, debido a una mala identificación de los accionistas o una mala búsqueda de requisitos. Es lo más difícil de corregir.

2. Una explosión de los requisitos derivados, los que surgen para satisfacer una cierta solución de diseño, por la complejidad del proceso de la solución, llegando a haber 50 veces más requisitos derivados que requisitos originales.

3. No detectar a tiempo ambigüedades entre requisitos. Si no se llegan a detectar, pueden causar defectos en el sistema que hagan que el software no llegue a usarse.

El cohete europeo Ariane 5 explotó debido a un desbordamiento al convertir un valor en coma flotante de doble precisión a un entero de 16 bits con signo, como resultado de mantener un requisito de software del Ariane 4 que no era necesario en el 5, relacionado con la velocidad horizontal y el ángulo de ataque detectados por un sensor de vuelo.

La NASA perdió el contacto con el Mars Polar Lander 10 minutos antes de su aterrizaje previsto debido a que el software malinterpretó señales que debía ignorar y creyó que uno de los brazos de la sonda había tocado suelo cuando estaba a 40 m de altura.

3.2. Fases

1. **Identificación y consenso.**

   a) **Contexto.** Se establecen las necesidades iniciales de la organización y se delimita el alcance del proyecto.

   b) **Trabajo preliminar.** Se identifican los interlocutores concretos y suministradores de información. Se hace un **estudio de viabilidad** para ver que el proyecto sea viable técnicamente con el presupuesto y el tiempo dados, y se decide entre crear el producto, comprarlo o subcontratar el desarrollo.

   c) **Adquisición.** Se establecen procedimientos de compra, adquisición y contratación si es necesario, y se prepara la adquisición y recogida de información.

2. **Representación y modelado.** Se escogen técnicas de representación y modelado apropiadas y se construyen modelos que reflejen las necesidades de los usuarios. Las técnicas de modelado (lenguajes y notaciones) se eligen según su **poder expresivo**: la variedad de tipos de requisitos que pueden ser capturados y la capacidad para suministrar mucha información en poco espacio.

   Se decide si usar **ingeniería inversa**, obtención de información a partir de un producto accesible al publico para determinar su composición, cómo funciona y cómo fue fabricado.

3. **Análisis.** Se **validan** los modelos construidos con los requisitos y se **verifica** la corrección interna y externa de los modelos, usando prototipos, inspecciones y revisiones.

4. **Documentación y comunicación.** Se organiza la información.

5. **Gestión y mantenimiento.** Se sigue la vida de los requisitos por todo el proceso de desarrollo para conseguir **trazabilidad**. Se definen los procedimientos de **gestión de cambios de requisitos**, manteniendo la integridad de los mismos durante la evolución del sistema y gestionando y documentando los cambios. Estos procedimientos deben aplicarse siempre tras la aprobación de los requisitos.

También se definen métricas sobre los productos del proceso de desarrollo, especificando pruebas, lo que permite estimar el coste de desarrollo y los recursos necesarios y refinar la planificación.

3.3. Análisis de requisitos

El ISO/IEC/IEEE 24765:2017 lo define como un «proceso de estudio de las necesidades del usuario para llegar a una definición de los requisitos del sistema, hardware o software» o un «proceso de estudio y perfeccionamiento de los requisitos del sistema, hardware o software».

Una **revisión de requisitos** es un proceso en que se presentan requisitos a partes interesadas para su aprobación o retroalimentación, o bien es un grupo de revisores que buscan errores, contradicciones, descripciones ambiguas, etc.

Si hay conflictos de intereses entre las partes, se hace una **negociación de requisitos**, en la que se detectan y resuelven los posibles conflictos; se definen de forma más precisa los limites del sistema y las interacciones de este con el entorno, y se eliminan redundancias, inconsistencias y puntos de desencuentro.

No es viable que los requisitos queden «congelados», pero tampoco lo es que cambien continuamente, por lo que en cierto momento se hace una ***sign-off***, una aprobación de ciertos requisitos por el cliente. Una ***baseline*** es un conjunto de requisitos identificados y acordados en un instante de tiempo, normalmente tras la *sign-off*.

3.4. Especificación de requisitos

Una **especificación** es un elemento de información que identifica de forma completa, precisa y verificable unas características esperadas de un sistema, servicio o proceso. En el análisis de requisitos se elabora un **documento de especificación de requisitos del software** (**SRS**, *Software Requirements Specification*) y, en su caso, uno de **requisitos del sistema** (**SyRS**, *System Requirements Specification*).

Un buen SRS documenta todo lo que el software debe hacer y lo que no mediante requisitos esenciales (funcionales, de rendimiento, restricciones de diseño y atributos) del software y de sus interfaces externas. También se puede elaborar una

Una buena SRS establece la base para el acuerdo entre clientes y proveedores, la estimación de costes, la planificación y futuras mejoras; reduce el esfuerzo de desarrollo al evitar re-diseñar o re-programar; sirve de referencia para la verificación y validación (V&V) y para comprobar el grado de cumplimiento del contrato, y facilita adaptar el sistema a nuevos usuarios y máquinas.

Capítulo 4

Documento de requisitos

Un **documento de requisitos** (**del proyecto**) o **especificación funcional** combina un SRS y un SyRS. Contiene un conjunto exhaustivo y preciso de los requisitos, validados, y sirve como contrato entre el cliente y los desarrolladores.

Se prepara en un bucle de 4 fases:

1. **Obtención**, en que se determinan los requisitos para llegar un conocimiento suficiente del problema y establecer los límites del sistema.
2. **Análisis**, en que se examinan los requisitos para delimitarlos y definirlos exactamente.
3. **Especificación**, en que se escriben los requisitos.
4. **Validación**, en que se examina si los documentos de requisitos definen el software que los usuarios esperan.

Los requisitos se pueden definir mediante:

- **Lenguaje natural.** Si el texto es preciso, es lo más fácil de entender, pero si la especificación es grande o compleja puede no ser manejable.
- **Texto y técnicas** intuitivas o formales, como diagramas, lenguajes de descripción de diseños o especificaciones lógicas formales. Es lo ideal para sistemas de tamaño medio o grande.
- **Solo técnicas.** Difícil de entender y rastrear.

4.1. Estándares

La NASA, la ESA y el Departamento de Defensa de EE.UU. tienen sus propios estándares para especificación de requisitos, pero los más usados son:

- ***IEEE Std. 830:1998, Recommended Practice for Software Engineering Requirements Specification*. IEEE Computer Society (1998).**

  Describe el contenido y las cualidades de un buen documento de SRS y presenta modelos para organizar requisitos específicos.

- ***ISO/IEC/IEEE 29148:2018, Systems and software engineering—Life cycle processes—Requirements engineering.***

  Suplanta al anterior. Incluye otros procesos y actividades relacionados con la ingeniería de requisitos, no solo la SRS, como la SyRS y la especificación de requisitos de *stakeholders* (StRS).

Según estos estándares, la SRS debe incluir:

- **Funcionalidad** del software.
- **Interfaces externas** con el usuario, el hardware y otras aplicaciones.
- **Rendimiento:** Velocidad, disponibilidad, tiempo de recuperación tras una caída del sistema, etc.
- **Atributos** de portabilidad, corrección, mantenimiento, seguridad, etc.
- **Restricciones de diseño:** Políticas sobre estándares, lenguajes de programación, entornos operativos, límites de recursos, etc.

Según el IEEE 830, la SRS no deben incluir el coste, los plazos de entrega, procesos de seguimiento, métodos de desarrollo, procedimientos generales de aseguramiento de calidad, criterios generales de validación y verificación ni procedimientos generales de aceptación. Tampoco se incluye el diseño.

4.2. Estructura

El tipo y número de documentos de requisitos, su organización y preparación y la forma de definir interrelaciones o jerarquías entre los requisitos deben estar claras desde el principio del proyecto. Se recomienda convertir esto en una norma en la organización, aunque con posibilidad de variación de un proyecto a otro según las características de cada uno.

Un documento de requisitos puede estructurarse como:

1. Necesidades del usuario o requisitos del negocio.
2. Documento de Requisitos del Producto (**PRD**, *Product Requirements Document*).
3. Especificación de requisitos de software (SRS) o hardware.
4. *Tests* para los requisitos de software (**STS**, *Software Test Specification*) o hardware.

Otra posible estructura es:

1. **Documento de Visión** (**y Alcance**), con la lista de participantes clave, los requisitos de negocio y el PRD.
2. **Documento de Casos de Uso**, con los requisitos de usuario como casos de uso.
3. SRS.

El IEEE 830 define la siguiente estructura para el SRS:

Tabla de Contenidos

1. Introducción
   1.1. Objetivo
   1.2. Ámbito
   1.3. Definiciones, siglas y abreviaturas
   1.4. Referencias
   1.5. Contenido *(explicación del contenido y organización del documento)*
2. Descripción general
   2.1. Perspectiva del producto
   2.2. Funciones del producto
   2.3. Características del usuario
   2.4. Limitaciones generales
   2.5. Supuestos y dependencias
3. Requisitos específicos

Apéndices

Índice

4.3. Clasificación de requisitos

Los requisitos pueden ser:

- **Del dominio del problema: requisitos de empresa** o **de producto**, **objetivos**, **necesidades** o **reglas de negocio**, y son requisitos de alto nivel de la organización o el cliente, recogidos en el PRD. Información general de las necesidades de los usuarios. Reglas.
- **Del dominio de la solución: Requisitos de usuario**, tareas que el usuario debe poder realizar con el producto, representadas mediante texto, casos de uso o descripciones de escenarios. **Requisitos de software**, funcionales o no, atributos de calidad y restricciones de diseño. Requisitos de hardware.

CMMi 2.0 distingue requisitos:

Del cliente Resultado de obtener necesidades; resolver conflictos entre ellas y las expectativas, limitaciones e interfaces, y definir las soluciones con las partes interesadas afectadas de forma aceptable para ellas.

Contractual Resultado del análisis y refinamiento de los requisitos del cliente en un conjunto de requisitos adecuado para su inclusión en solicitudes o acuerdos con proveedores.

De producto Refinamiento de los requisitos de cliente en lenguaje próximo a los desarrolladores para guiar el diseño del sistema.

De componente de producto Especificación completa de un componente de un producto o servicio, incluyendo ajuste, forma y función.

De contexto Estándares aplicables, leyes, políticas, prácticas habituales, decisiones de dirección, rendimiento, etc.

Derivados Inferidos de requisitos de contexto o necesarios para especificar un componente. Puede aparecer también en el análisis y diseño de componentes.

Asignado (*allocated*) Requisito resultante de imponer todo o parte de un requisito de nivel superior a un componente de diseño de nivel inferior, lógico o físico, incluyendo una persona, un consumible, un incremento de la entrega o la arquitectura.

Técnicos Propiedades de productos o servicios a ser adquiridos o desarrollados.

No técnicos Afectan a la adquisición o el desarrollo del sistema pero no son propiedades de este.

4.4. Propiedades de los requisitos

Un requisito puede ser:

- **Necesario** si define una capacidad o característica esencial, aplicable en la actualidad y no obsoleta, de modo que su eliminación produce una deficiencia que no puede ser compensada por otras capacidades del producto.

  Los requisitos que puedan tener capacidad o solo son aplicables en fechas concretas deben ser claramente identificados.

- **Independiente** (**de la implementación**) si no establece restricciones innecesarias sobre el diseño arquitectural.

- **Preciso**, **inequívoco** o **no ambiguo** si admite una única interpretación.

  La mayoría de defectos en los requisitos se deben a la ambigüedad del lenguaje, que puede surgir en la definición de nuevos requisitos, la incorporación de nuevas necesidades y los cambios en los requisitos durante el desarrollo.

  Para evitar esto debemos usar el lenguaje de forma precisa, pudiendo usar plantillas o estructuras fijas de lenguaje o técnicas de descripción rigurosas y estructuradas. Para resolver una ambigüedad existente, pedimos el significado correcto a quienes escribieron el requisito.

- **Consistente** si no entra en conflicto con otros requisitos.

- **Completo** si no necesita ampliarse para describir suficientemente la capacidad o característica requerida.

- **Singular** si no es des componible en varios requisitos y no usa conjunciones.

- **Factible** si es técnicamente alcanzable, no requiere grandes avances tecnológicos y cumple con las restricciones del proyecto con un riesgo aceptable.
- **Verificable** si hay una forma efectiva en tiempo y coste de determinar si el sistema satisface el requisito en grado suficiente para convencer a las partes relevantes.

Los requisitos se pueden relacionar por:

- **Trazabilidad inclusiva:** Un requisito destino depende de uno origen, de modo que si desaparece el origen, el destino no tiene sentido y debe desaparecer.
- **Jerarquía** o **refinamiento:** Un requisito padre es explicado con más detalle por sus hijos. Es un caso de trazabilidad inclusiva.
- **Exclusión:** La elección de un requisito impide la de otro incompatible.

Un requisito es **trazable hacia arriba** si enlaza a su fuente u origen, como una descripción documentada de una necesidad de *stakeholder*, un requisito de nivel superior, un plan de negocio, etc.; **trazable hacia abajo** si enlaza a requisitos de nivel inferior más refinados o a otros artefactos resultantes de añadir el requisito, y **trazable** si lo es hacia arriba y hacia abajo.

4.5. Meta-información

Las SRS suelen incluir, junto con los requisitos, meta-información sobre estos, en forma de atributos. El método VOLERE incluye, entre otros los siguientes atributos de cada requisito:

- **Identificador de requisito**, único en todo el proyecto incluso entre requisitos borrados.
- **Tipo de requisito**.
- **Razón** o justificación.
- **Fuente**, trazabilidad hacia arriba.
- **Satisfacción del cliente** si el requisito se implementa, del 1 al 5.
- **Insatisfacción del cliente** si el requisito no se implementa, del 1 al 5.
- **Dependencias**, trazabilidad hacia abajo.

4.6. Propiedades de las SRS

Un documento o grupo de requisitos puede ser:

- **Completo** si incluye todos los requisitos que los *stakeholders* esperan haber cumplido en la versión correspondiente del sistema o parte del sistema. Cuando un requisito o subgrupo no está completo, lo indicamos con una etiqueta como **TBD** (***To Be Determined***).

- **Consistente** si no contiene requisitos excluyentes ni duplicados, no se usan varios términos para el mismo concepto y su satisfacción no entra en conflicto con la de otro documento previamente aprobado.
- **Alcanzable** si se puede construir un sistema que satisfaga los requisitos.
- **Asequible** si se puede construir dicho sistema dentro del coste, la planificación, la tecnología disponible y la ley.
- **Limitado** si no extiende el alcance de la solución más de los necesario para satisfacer las necesidades de los usuarios.

Según el IEEE 29148, un documento o grupo de requisitos debe ser completo, consistente, asequible y limitado, y sus requisitos deben ser necesarios, independientes de la implementación, no ambiguos, consistentes, completos, singulares, factibles, trazables y verificables.

El no cumplimiento de alguna propiedad indica posibles problemas en los requisitos. Se puede crear una **lista de comprobación** con propiedades de requisitos y de grupos de requisitos para asegurarse en la **revisión de requisitos** de que estas se cumplen.

4.7. Gestión del cambio

En el desarrollo o el mantenimiento, se puede cambiar el documento de requisitos rellenando una **plantilla de solicitud de cambio** y enviándola al **grupo de gestión de cambios**, un grupo idealmente de entre 3 y 5 personas, normalmente responsables de la construcción del documento de requisitos, que se encarga de estudiar los posibles cambios.

Una plantilla podría contener como campos «Petición de cambio» (nombre del proyecto afectado), «Identificador del requisito afectado» (documento y código), «Nombre asignado al cambio», «Descripción» (del cambio), «Pedido por» (nombre y cargo) y «Justificación».

4.8. Problemas

Manejar documentos de requisitos puede ser complicado por la gran cantidad de requisitos, por lo que hay que obviar requisitos obvios y no especificar absolutamente todo.

El nivel de detalle en que debemos expresar los requisitos depende del nivel de abstracción en que nos encontremos, es decir, de si los requisitos son de empresa, de usuario o de software, por lo que es importante la trazabilidad para ligar niveles de abstracción.

Finalmente, está el problema de que a veces un requisito pertenece a varias categorías y puede ser asignado a una o más secciones del documento de requisitos. Una solución es usar referencias cruzadas.

Chapter 5

Técnicas de obtención de requisitos

La **obtención de requisitos** consiste en determinar los requisitos del sistema a desarrollar a partir de los objetivos generales del software, el dominio del problema, los involucrados en el proceso, el entorno de operación, el entorno de la organización, etc., mediante entrevistas, escenarios, prototipos, reuniones de grupo, observación, etc., para llegar a un conocimiento suficiente del problema a resolver.

5.1. Técnicas intuitivas

5.1.1. Entrevistas

Constan de:

1. **Planificación.** Se establece qué datos obtener de qué personas, cuándo y dónde.
2. **Preparación.** Se obtiene información externa útil para elaborar preguntas guía.
3. **Inicio.** Se expone cómo se hará la entrevista, los objetivos y la duración.
4. **Desarrollo.** Se hacen las preguntas y se recaba información.
5. **Cierre.** Se resume la información para asegurarnos de que tenemos la que queríamos.
6. **Conclusiones.** Se resume formalmente la entrevista, incluyendo conclusiones a confirmar en otras entrevistas.

Se puede preguntar sobre detalles específicos, sobre la visión de futuro que el entrevistado tiene de algo, sobre ideas alternativas, sobre una solución mínimamente aceptable y sobre otras fuentes de información.

Al haber pocas personas con mucha información sobre el sistema a construir, la relación es más personal y directa, pero se requiere mucho tiempo y es más difícil detectar errores ya que solo en analista revisa los resultados.

5.1.2. FAST

Las ***Facilitated Application Specification Techniques*** son técnicas de obtención de requisitos orientadas a equipos de trabajo.

El equipo escribe un **documento de solicitud de producto** de 1–2 hojas. Entonces se establece una fecha, hora y lugar para la próxima reunión y se elige a los asistentes entre usuarios finales, clientes y productores, a los que se les hace llegar la solicitud de producto y se les pide que preparen:

- Una **lista de objetos** que forman parte del entorno del nuevo producto.
- Una **lista de servicios** (procesos o funciones) que manipulen o interactúen con los objetos.
- Una **lista de restricciones** de coste, tamaño, rendimiento y reglas de negocio.

Se informa a los asistentes de que las listas no tienen que ser exhaustivas, pero que tienen que reflejar su perspectiva del sistema. Además, se elige a un cliente o productor como moderador de la reunión.

JAD (***Joint Application Development***) es una FAST que, basándose en que quien mejor conoce un trabajo es quien lo desempeña, quien mejor conoce las posibilidades del software es el equipo de desarrollo y los sistemas de información y procesos de negocio trascienden las fronteras de un único departamento o sección, busca que los participantes en los grupos que diseñan los sistemas trabajen de igual a igual.

Se hacen reuniones grupales entre usuarios y analistas en 2–4 días, con 2 horas por reunión, con ayudas visuales para mejorar la comunicación y trabajando directamente sobre el documento de requisitos a generar. Se empieza con un documento de trabajo y se discute hasta aprobar el documento de requisitos al final.

La participación de los usuarios con los JAD es mayor que con las entrevistas, la satisfacción es mayor y se ahorra tiempo, pero los JAD son más difíciles de llevar a la práctica ya que hace falta más organización para gestionar reuniones de más personas, entre 5 y 25.

5.1.3. Casos de uso

Son descripciones genéricas de uso de un sistema por parte de un actor para conseguir resultados u objetivos, y representan parte de, uno o varios requisitos funcionales. Se representan como en PDS.

Pueden usarse como ayuda para descubrir requisitos que luego se convertirán en requisitos textuales. Los documentos de requisitos pueden contener diagramas de casos de uso como descripción gráfica complementaria, y debe documentarse la trazabilidad entre requisitos y casos de uso.

Los casos de uso pueden dirigir el desarrollo, expresándose en términos de un modelo de objetos del dominio para luego estructurarse con un modelo de análisis y realizarse en un modelo de diseño, que se implementa en un modelo de implementación (código) que se prueba en un modelo de pruebas (*tests*).

5.1.4. Escenarios

Un **escenario** o **instancia de caso de uso** es una secuencia específica de acciones e interacciones concretas entre los actores y el sistema. Consta de:

1. Descripción de lo que el sistema y los usuarios esperan al principio del escenario.
2. Descripción del flujo normal de eventos en el escenario.
3. Descripción de lo que puede fallar y cómo se maneja esto.
4. Información de otras actividades que pueden ocurrir a la vez.
5. Descripción del estado del sistema cuando el escenario termina.

Un caso de uso puede definirse como un conjunto de escenarios de éxito y fallos relacionados en los que los actores usan un sistema para alcanzar un mismo objetivo o resultado.

5.2. Técnicas formales

El ISO/IEC/IEEE 24765:2017 define **especificación formal** como:

1. Especificación usada para probar matemáticamente la validez de una implementación o derivar matemáticamente la implementación.
2. Especificación escrita en una notación formal, a menudo para ser usada en pruebas de corrección.
3. Especificación escrita y aprobada de acuerdo con algunos estándares establecidos.

Los **modos** o **niveles de aplicación** de las especificaciones formales se refieren a que estas se pueden aplicar o no para generar ejecutables, y para demostrar propiedades, manualmente o con pruebas generadas automáticamente.

5.2.1. Notaciones formales

Una **notación formal** está formada por:

1. Una **sintaxis**, un método de decisión de la corrección gramatical de sus representaciones.
2. Una **semántica**, una serie de reglas para interpretar las representaciones de forma precisa y significativa en el dominio considerado.
3. Una **teoría de prueba**, una serie de reglas para inferir información útil a partir de la especificación.

Hay dos enfoques principales para notaciones formales:

- **Algebraico:** Se describe el sistema en función de operaciones y sus relaciones. Las interfaces se suelen describir en las siguientes secciones:
  - **Introducción,** con los ***sorts*** o nombres de tipos de la entidad que se está especificando.

- **Descripción** informal de las operaciones.
- **Signatura,** sintaxis de la interfaz para el tipo de datos.
- **Axiomas,** que caracterizan la semántica de las operaciones.

- **Basado en modelos:** Se construye un modelo del sistema mediante construcciones matemáticas, y se definen las operaciones según cómo modifican el estado del sistema.

También diferenciamos entre notaciones formales secuenciales y concurrentes.

Z es una notación formal secuencial basada en modelos, formada por:

- Una notación para **esquemas**, formada por:
  - Nombre del esquema.
  - **Signatura**, con líneas en formato *atributo*[, ...]: *Tipo*.
  - **Predicado**, con líneas que especifican invariantes entre los atributos.
- Una notación para **operaciones** sobre los esquemas.

Maude es una notación formal algebraica y concurrente que soporta módulos funcionales, de sistema y orientados a objetos. Puede usarse para especificar aplicaciones y sistemas generales.

5.2.2. Métodos formales

Una **técnica formal** es una que usa especificaciones formales, y un **método formal** es una técnica formal con un procedimiento de uso.

Según Hall, los principales **mitos** sobre los métodos formales son que:

1. Pueden garantizar que el software sea perfecto.
2. Tratan sobre pruebas de programas.
3. Solo son útiles para sistemas de seguridad crítica.
4. Requieren matemáticos muy entrenados en su uso.
5. Incrementan el coste de desarrollo.
6. Son inaceptables para el usuario.
7. No se usan en la práctica.

Las técnicas formales introducen rigor en la documentación, evitando la ambigüedad al usarse como contrato entre clientes y proveedores, y ayudan a comprender propiedades del software a construir. Además, facilitan la implementación, que se puede hacer automáticamente, y la prueba, que puede ser formal.

Por otro lado, muchas veces el aspecto más crítico no es la calidad sino la rapidez en sacar el producto al mercado, y las técnicas formales son difíciles de aplicar y sistemas grandes y no son apropiadas para especificar interfaces de usuario o interacciones entre usuarios.

Los métodos formales se usan en sistemas críticos, sistemas reproducidos muchas veces, sistemas empotrados en hardware, sistemas de alta calidad por requerimientos comerciales y sistemas que va a usar mucha gente.

5.3. Herramientas CARE

Las **herramientas CARE** (*Computer-Aided Requirements Engineering*) son herramientas que facilitan la definición, la organización, el almacenamiento y la gestión de los requisitos. Suelen incluir:

- Informes de alto nivel sobre los requisitos.
- Plantillas para realizar la documentación del proyecto.
- Integración con otras herramientas.
- Importación y exportación de requisitos, por ejemplo con el formato ReqIF estandarizado por la OMG, un estándar que los apuntes dicen falsamente que es de facto.[1]

Algunas herramientas CARE son:

- El software privativo ReqView.
- El software privativo DOORS.
- El software privativo Borland Caliber-RM.
- El software privativo TRC Systems Engineering Suite, que usa heurísticas para valorar la corrección, consistencia y completitud de los requisitos y ayudar a su calidad y soporta reutilización de requisitos con un motor de búsqueda semántico.
- El software privativo Prover, basado en técnicas formales, que permite probar la seguridad de un sistema y otras propiedades matemáticamente.

Algunas herramientas permiten añadir *plug-ins* o *add-ins* que aumentan su funcionalidad o las conectan con otras herramientas.

[1] Tienen una definición de «de facto» distinta a la definición estándar de facto (la que todo el mundo entiende) y de jure (la de la RAE, que es la misma).

Capítulo 6

Métodos de la ingeniería de requisitos

6.1. Enfoque orientado a objetivos

Un **objetivo** es una meta que el sistema debería alcanzar. Puede ser funcional o no funcional, y se puede formular a distintos niveles de abstracción (estratégicos o técnicos). Un **agente** o **componente activo** del sistema es un componente con comportamiento asociado, como una persona, el software o ciertos instrumentos o hardware. Un **componente pasivo** es uno sin comportamiento asociado.

Normalmente se necesitan varios agentes para realizar un objetivo. Cuando un objetivo lo puede realizar un solo agente o tipo de agente, se llama **requisito** si el agente está dentro del sistema.

Los objetivos pueden representarse de manera informal pero precisa; semiformal mediante texto o gráficos, y formal mediante una lógica temporal, siendo esta última forma opcional y reservada para los aspectos más críticos.

KAOS es un método orientado a objetivos que proporciona un lenguaje de especificación de requisitos y los elementos necesarios para describir conceptos de este enfoque. Incluye 4 modelos elaborados incrementalmente:

1. Modelo de objetivos (***goal model***).
2. Modelo de objetos (***object model***).
3. Modelo de responsabilidades de agentes (***agent responsibility model***).
4. Modelo de operaciones (***operation model***).

En un modelo de objetivos, los objetivos abstractos (*soft goals*), no funcionales, se representan con una nube, y los objetivos formalizables se representan con un romboide que es como un rectángulo con la parte de arriba desplazada a la derecha. Una flecha de un objetivo a otro indica que el primero influye en el segundo, contribuyendo fuertemente si la flecha se etiqueta, cerca del inicio, con «++» y entrando en conflicto fuertemente si se etiqueta con «--».

Dos o más flechas dirigiéndose a un punto (círculo pequeño) y una flecha partiendo de este indican una conjunción, y dos o más flechas hacia el mismo objetivo que se unen por la parte del final por segmentos paralelos es una disyunción.

A un objetivo formalizable se le puede dar una descripción formal como sigue:

Goal Nombre del objetivo

InformalDef Descripción textual

FormalDef Descripción en notación matemática.

6.2. Enfoque basado en puntos de vista

Un . La construcción de un sistema de información complejo implica la participación de muchos *stakeholders*, cada uno con una visión del sistema que suele ser parcial o incompleta, ligada a su papel.

En esta situación puede ser adecuado un enfoque basado en **puntos de vista** (***viewpoints***), combinaciones de un agente con su visión del sistema. En este integramos los puntos de vista de los agentes para derivar la información completa de los requisitos.

Un punto de vista se considera como un receptor de servicios del sistema y un emisor de datos e información de control. Los servicios, similares a casos de uso, se describen con requisitos funcionales y las restricciones con requisitos no funcionales.

VORD (*Viewpoint-Oriented Requirements Definition*) es una técnica basada en puntos de vista con las siguientes actividades:

1. **Identificación de puntos de vista.** Se descubren los puntos de vista que reciben servicios y se identifican los servicios proporcionados a cada uno.

2. **Estructuración de puntos de vista.** Se agrupan puntos de vista relacionados en una jerarquía. Los servicios comunes se proporcionan a niveles más altos de la jerarquía.

   Por ejemplo, en un banco, solo los clientes registrados pueden sacar dinero, pero tanto estos como los clientes casuales pueden querer pagar recibos, por lo que este último servicio se trasladaría a un punto de vista «cliente» del que heredan «cliente registrado» y «cliente casual».

3. **Documentación de puntos de vista.** Se refina la descripción de los puntos de vista y servicios identificados.

4. **Mapeo entre puntos de vista y sistema.** Se transforma el análisis de los puntos de vista en un diseño.

Tras cada fase se puede volver a cualquiera de las anteriores si hay un error. Un punto de vista se puede definir como sigue:

Referencia Nombre del rol.

Atributos Lista de atributos relevantes del agente al interactuar con el sistema.

Eventos Lista de acciones que puede iniciar en el sistema, similar a las operaciones de los DSS.

Servicios Lista de referencias a los servicios que usa.

Subpuntos de vista Lista de puntos de vista subordinados en la jerarquía.

Un servicio se puede definir con la siguiente plantilla:

Referencia Nombre del servicio.

Razón Objetivo conseguido al implementarlo.

Especificación En qué consiste y cómo se usa.

Puntos de vista Lista de referencias a los puntos de vista que lo usan.

Requisitos no funcionales

Proveedor

Otras técnicas basadas en puntos de vista son **VOSE** (*Viewpoint-Oriented Software Engineering*) y PREview.

6.3. Reutilización de requisitos

Reutilizar requisitos puede mejorar la calidad, la productividad y, al evitar trabajo, la sostenibilidad medioambiental. Es ventajoso entre aplicaciones de un mismo dominio, pues muchos requisitos expresan restricciones sobre el sistema o su operación derivadas del dominio de aplicación.

También lo es en requisitos relacionados con el estilo de presentación de la información, evitando re-diseñar la presentación de la aplicación y dando un estilo consistente, y en requisitos relacionados con políticas de empresa o normas legales, como políticas de seguridad, leyes de proyección de datos, etc.

6.3.1. SIREN

SImple REuse of requiremeNts[1] es un método práctico y sencillo de reutilización de requisitos compatible con estándares ISW y técnicas usadas en la industria, basado en requisitos en lenguaje natural con un conjunto mínimo de atributos y un modelo de trazabilidad. Introduce el concepto de **repositorio** de requisitos, una colección de **catálogos de requisitos reutilizables** por tema. Existen herramientas de soporte para SIREN como el software privativo SirenTool o Pantalasa.

SIREN se puede usar **p**ara reutilización (**SIRENp**), creando requisitos para su uso posterior, o **c**on reutilización, generalizando requisitos de proyectos una vez estos se validan (**SIRENc**). En SIRENc:

1. Se seleccionan requisitos añadiendo requisitos del catálogo a plantillas vacías, o a la iteración anterior del análisis de requisitos.

[1] Se ve que esto lo diseñó un cani.

2. Se integran y negocian requisitos resultado de un análisis de requisitos.

3. Los requisitos se documentan y validan.

4. El resultado se usa para mejorar el repositorio, para las siguientes fases del desarrollo y para los siguientes incrementos del sistema.

Una «jerarquía» para el documento de requisitos con SIRENc puede tener una SyRS según el IEEE 1233 y el 12207.1, una **SyTS** (*System Test Specification*, especificación de pruebas del sistema), una STS, un SRS según IEEE 830 y una **IRS** (*Interface Requirement Specification*, especificación de requisitos de interfaz) según el IEEE 830.

Los requisitos SIREN pueden estar **parametrizados**, con partes entre corchetes como por ejemplo «[n, n >= 6]» para indicar un número $n \geq 6$.

6.3.2. Otros métodos

Un **patrón de requisitos** es un requisito escrito a muy alto nivel de abstracción, muy genérico y relativamente fácil de adaptar.

El **análisis de dominio** consiste en investigar, capturar y especificar conocimiento genérico sobre alguna tarea específica. No se enfoca a un solo proyecto, sino que busca definiciones generales que puedan ser reutilizadas en muchos proyectos.

6.3.3. Uso de reutilización

Las técnicas de reutilización pueden constituir de por sí un proceso de ingeniería de requisitos o ser integradas con otras técnicas.

En 2012, Chernak hizo un estudio sobre el uso de reutilización de requisitos.

Según este, las estrategias de ingeniería de requisitos más usadas son las tradicionales, seguidas por las dirigidas por casos de uso y las ágiles. De estas, las que usan tradicionales son las que más usan reutilización y las que usan ágiles las que menos.

Los obstáculos más importantes a la reutilización eran que los requisitos desarrollados en versiones anteriores eran incompletos, o incluso inexistentes o no disponibles, y no se podían reutilizar; que estaban pobremente estructurados y era difícil identificar cuáles podían reutilizarse, y que no se mantenían actualizados.

Los principales beneficios de la reutilización identificados fueron la reducción del tiempo de puesta en el mercado y la del coste de desarrollo.

6.4. Objetivos de desarrollo sostenible[2]

El manifiesto de Karlskrona considera las siguientes dimensiones en la ingeniería de software:

- **Individual.** Bienestar físico y mental de las personas, educación, respeto a uno mismo, habilidades, movilidad, etc.

- **Social.** Comunidades sociales y los factores que erosionan la confianza en la sociedad. Incluye igualdad social, justicia, empleo, democracia, etc.

[2]Incluidos con calzador y sin ninguna maña por los profesores de la asignatura. ¡Disfrute!

- **Económica.** Activos y valor añadido. Incluye creación de riqueza, prosperidad, rentabilidad, inversión, etc.
- **Técnica.** Longevidad de la información, los sistemas y la infraestructura y su evolución y adaptación a cambios del entorno. Incluye mantenimiento, innovación, obsolescencia, integridad de los datos, etc.
- **Medioambiental.** Efectos a largo plazo de las actividades humanas en la naturaleza. Incluye ecosistemas, recursos naturales, cambio climático, producción de alimentos, agua, contaminación, desperdicio, etc.

Considera que solo se tienen en cuenta los requisitos técnicos y se deberían considerar todos. Distinguimos entre ***Green in IT***, o el uso de buenas prácticas para el desarrollo sostenible de software, y ***Green by IT***, o la inclusión de requisitos sostenibles, por ejemplo en un catálogo de requisitos.

6.5. Enfoque i*

A diferencia de otros métodos, este tiene una fase previa a la especificación de requisitos, en la que se indica cómo el sistema debe satisfacer los objetivos de la organización, qué alternativas puede haber, cuáles de esas alternativas afectan a varios implicados y cómo se podrían resolver los intereses de los implicados identificando posibles conflictos entre ellos. Forma parte del estándar internacional ITU-T Z.151, *User Requirements Notation* (**URN**), de 2008.

El i* define dos modelos:

- **Modelo de dependencia estratégica** (**SD**, *Strategic Dependency*). Modela dependencia entre actores debidas a metas, metas suaves, tareas o recursos, dando una visión global del sistema.
- **Modelo de razonamiento estratégico** (**SR**, *Strategic Rationale*). Modela el razonamiento estratégico interno de un actor.

Un diagrama SR está formado por regiones del plano que representan actores y contienen un círculo con el nombre del actor, y también pueden contener otros elementos con su nombre dentro.

Estos elementos son metas, representadas con un rectángulo redondeado; metas suaves, como si a una meta le hubiera caído un objeto grande encima y la hubiera deformado; tareas, como un hexágono regular en horizontal tras alargar sus lados horizontales, y recursos, representados con rectángulos. Pueden estar dentro de un actor o fuera.

Una flecha de una tarea a una meta con la punta de flecha en medio de la línea es un enlace de medios a fines, y si la punta de flecha tiene volumen y la flecha termina en una meta suave, es un enlace de contribución a meta suave.

Una línea de una tarea a otro elemento de cualquier tipo (salvo actor) y con un pequeño segmento cruzado en el lado de la tarea es un enlace de descomposición de tarea, y la tarea se descompone en los elementos que parten de esta por enlaces de este tipo. Un diagrama SD es similar pero sin mostrar los objetivos y tareas internos de los actores.

6.6. MAGERIT

Es un método formal usado en la Administración Pública española para investigar los riesgos de seguridad de sistemas y recomendar medidas apropiadas para controlar estos riesgos.

Sus objetivos son concienciar a los responsables de los sistemas de información de la existencia de riesgos y la necesidad de atajarlos a tiempo y ofrecer un método sistemático para analizar tales riesgos, ayudando a descubrir y planificar las medidas oportunas para controlar los riesgos y preparando a la organización para procesos de evaluación, auditoría, certificación o acreditación.

En el Ciclo de Gestión de la Seguridad, se crea una Política de Seguridad de los Sistemas de Información y una Organización de Seguridad de los Sistemas de Información, a partir de los que se hace un Análisis y Gestión de Riesgos con MAGERIT y, como resultado, se crea un Plan de Seguridad de los Sistema de Información, que se somete a Implantación y Seguimiento.

Los elementos de MAGERIT son:

- **Activos:** recursos del sistema o relacionados con este.
- **Amenazas:** eventos que pueden desencadenar un incidente.
- **Vulnerabilidades:** Estimación de la exposición efectiva de un activo a una amenaza.
- **Impacto:** Consecuencia que tiene la ocurrencia de una amenaza sobre un activo.
- **Riesgo:** Estimación del grado de exposición a que se dé una amenaza.
- **Salvaguarda:** Procedimiento o mecanismo tecnológico para reducir el riesgo.

La versión 3 de MAGERIT incluye el software privativo PILAR.

Capítulo 7

Gestión de proyectos

Las organizaciones logran objetivos realizando trabajos, que pueden ser:

- **Trabajos operativos** u **operaciones:** Esfuerzos permanentes que producen salidas repetitivas, con recursos asignados para realizar unas mismas tareas.
- **Proyectos:** Esfuerzos temporales para crear un producto, servicio o resultado único.

ITIL (*Information Technology Infrastructure Library*) es una metodología de gestión de operaciones que transforman entradas en salidas para la producción continua de bienes o servicios, y busca que las operaciones de negocio se desarrollen eficientemente y con los recursos óptimos para cumplir con la demanda de los clientes.

La **dirección de proyectos** es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades de proyecto para cumplir con sus requisitos, e incluye habilidades como la identificación de requisitos; la comunicación activa con los interesados; el equilibrio de demandas entre calidad, alcance, tiempo y costes; la gestión de recursos, y la adaptación del plan y el enfoque según las inquietudes y las expectativas.

Una dirección de proyectos eficaz supone cumplir los objetivos de negocio; satisfacer las expectativas de los interesados; ser predecibles; aumentar las posibilidades de éxito; entregar productos adecuados en el momento adecuado; resolver problemas e incidentes; responder a los riesgos; optimizar el uso de recursos; identificar, recuperar y concluir proyectos fallidos; gestionar restricciones de alcance, calidad, cronograma, costos y recursos, y gestionar el cambio.

7.1. Planificación estratégica

Un **programa** es un grupo de proyectos relacionados, con dirección coordinada para obtener control y beneficios.

Un **plan estratégico** busca aprovechar oportunidades de negocio alineadas con metas estratégicas. Un **portafolio** es una agrupación de proyectos, programas y otros trabajos, no necesariamente relacionados, hecha para facilitar la gestión y cumplir con los objetivos estratégicos del negocio.

Un **producto entregable** es un producto de trabajo que se puede medir y verificar, y puede corresponder al proceso de dirección de proyectos o ser productos finales o componentes de los

mismos. Para facilitar la gestión de un proyecto, su ciclo de vida se divide en **fases**, conjuntos de actividades relacionadas que culminan con la terminación de uno o más entregables.

Podemos dividir un portafolio en **subproyectos**, componentes más fáciles de gestionar, basándonos en la fase del ciclo de vida, las habilidades que se requieren o la tecnología usada.

Una **oficina de dirección de proyectos** (**PMO**, *Project Management Office*) es una unidad para centralizar y coordinar la dirección de proyectos. Supervisa la dirección de proyectos o programas buscando la planificación coordinada, la priorización y proyectos vinculados a objetivos de negocio generales. Asume funciones de respaldo a la dirección y de dirección de proyectos.

Tiene autoridad para actuar como interesada integral, tomar decisiones clave al comienzo de cada proyecto, hacer sugerencias, concluir proyectos y participar en la selección y gestión de recursos de proyectos compartidos o dedicados.

Los **interesados** en un proyecto son personas y organizaciones que participan activamente en el proyecto o cuyos intereses pueden verse afectados como resultado del proyecto. Incluyen al equipo del proyecto, los clientes y la PMO.

Los **factores ambientales** son condiciones que el equipo del proyecto no puede controlar pero influyen en el proyecto. Hay factores **internos**, como las normas culturales, la estructura, la distribución geográfica, la infraestructura, el software, la disponibilidad de recursos y la capacidad de los empleados, y factores **externos**, como las condiciones de mercado, las restricciones legales, las influencias sociales y culturales, las bases de datos comerciales y los estándares de la industria.

7.2. Sistemas de gestión de proyectos

Los **activos de los procesos de la organización** son planes, procesos, políticas, procedimientos y bases de conocimiento usados por la organización.

Un **sistema de gestión de proyectos** es un conjunto de recursos, técnicas y metodologías usados para gestionar un proyecto, cuya forma de uso se describe en un **plan para la dirección del proyecto**.

Hay varias formas de organizar proyectos, desde la funcional, más libre, hasta la orientada a proyectos, con más carga de gestión. La matricial tiene carga intermedia.

Las **organizaciones basadas en proyectos** son las que obtienen sus ingresos de ejecutar proyectos para otros, como las consultoras, y han adoptado dirección por proyectos. Suelen tener sistemas de gestión para facilitar la dirección de proyectos, como sistemas financieros, y una PMO.

Las organizaciones no basadas en proyectos no suelen tener sistemas de gestión para respaldar eficientemente las necesidades de proyectos, lo que dificulta la dirección, aunque algunas tienen departamentos u otras subunidades que operan como organizaciones basadas en proyectos.

Un enfoque **prescriptivo**, **descriptivo** o **disciplinado** a la dirección de proyectos, como PRINCE2, ISO 21500:2012 o PMBOK, es uno que planifica y predice en detalle las actividades y la asignación de recursos, con muchos artefactos creados en un ambiente burocrático.

Un enfoque **ágil** o **adaptativo** es uno que acepta el cambio como inevitable y fomenta la adaptación y hacer solo lo imprescindible para generar valor. Los métodos **híbridos** son los que gestionan el proyecto con un marco conceptual prescriptivo pero aplican prácticas ágiles, y son muy usados por la industria.

Los métodos prescriptivos son más adecuados para proyectos grandes o complejos, mientras los ágiles lo son más para proyectos pequeños o sencillos.

7.3. PMBOK

PMBOK (*Project Management Body of Knowledge*) es un conjunto de fundamentos de dirección de proyectos generalmente aceptados como buenas prácticas y un esquema de trabajo para gestionar cada aspecto de un proyecto y definir metodologías.

Se define por la **Guía de los Fundamentos para la Dirección de Proyectos** o **Guía del PMBOK**, elaborada por el **PMI** (*Project Management Institute*), institución reconocida por ANSI e IEEE como organización desarrolladora de estándares, y su 6ª edición es ANSI/PMI 99-001:2017.

PMI aporta certificaciones profesionales como de **PMP** (*Project Management Professional*), **CAPM** (*Certified Associate in Project Management*) y **PMI-ACP** (*PMI Agile Certified Practitioner*).

PMBOK se estructura en:

- Un ciclo de vida de 4 fases: inicio del proyecto; organización y preparación; ejecución del trabajo, y finalizar el proyecto.
- 5 grupos de procesos de dirección de proyectos que se pueden aplicar a todas las fases del ciclo de vida:
  - **Procesos de inicio:** Definen y autorizan el proyecto o una fase.
  - **Procesos de planificación:** Definen y refinan los objetivos y planifican el curso de acción requerido para lograr los objetivos, incluyendo el alcance pretendido.
  - **Procesos de ejecución:** Integran a personas y recursos para ejecutar el plan de gestión del proyecto.
  - **Procesos de seguimiento y control:** Miden y supervisan regularmente el avance, identificando variaciones y tomando medidas correctivas.
  - **Procesos de cierre:** Formalizan la aceptación del producto, servicio o resultado y terminan ordenadamente el proyecto o fase.
- 10 áreas de conocimiento que se pueden aplicar a todos los grupos de procesos.

Un **director de proyecto** es una persona asignada por la organización para liderar al equipo que desarrolla el proyecto y hacerse responsable de su éxito o fracaso. El equipo de dirección debe tener las siguientes áreas de experiencia:

- **Fundamentos de la dirección de proyectos.** Conocimientos propios del campo de la dirección de proyectos, como los de la Guía del PMBOK.
- **Conocimientos, normas y regulaciones del área de aplicación.** Las áreas son categorías de proyectos con elementos significativos comunes como elementos técnicos, departamentos funcionales y disciplinas de respaldo.

- **Comprensión del entorno del proyecto.** Los contextos se dan en un contexto social, económico y ambiental dado por el entorno cultural, social, internacional, político y físico, y tienen impactos positivos y negativos, deseados y no deseados.
- **Conocimiento y habilidades de dirección general.** Planificación estratégica, táctica y operativa; organización; selección del personal; ejecución y control de las operaciones; gestión financiera; contabilidad; compras; contratos; derecho mercantil; logística; fabricación; distribución, etc.
- **Habilidades interpersonales.** Liderazgo, comunicación, negociación, resolución de problemas, interacción proactiva, influencia, motivación, formación, etc.

Un **acta de constitución del proyecto**, **mapa de proyecto**, **mapa de constitución** o ***project charter*** es un documento que ofrece una visión global del proyecto, incluyendo:

- **Visión:** propósito; objetivos específicos y medibles, y alcance.
- **Organización:** clientes, *stakeholders*, roles y responsabilidades.
- **Implementación:** plan general de ejecución, con fases, hitos, tiempos, actividades, ruta crítica, dependencias, recursos, etc.
- **Riesgos:** evaluar principales riesgos, probabilidad de ocurrencia y mitigaciones.

7.4. Áreas de conocimiento en PMBOK

7.4.1. Gestión de las Comunicaciones del Proyecto

Se asegura de que se satisfacen las necesidades de información del proyecto y sus interesados, e incluye definir estrategias según las necesidades y transmitir mensajes según la estrategia. Requiere habilidades como:

- Escuchar activamente.
- Formular preguntas para mejorar la comprensión.
- Educar para aumentar el conocimiento del equipo.
- Investigar para identificar o confirmar información.
- Identificar y gestionar expectativas.
- Persuadir a una persona u organización para llevar a cabo una acción.
- Negociar para lograr acuerdos entre partes.
- Resolver conflictos para prevenir impactos negativos.
- Resumir, recapitular e identificar las próximas etapas.

La comunicación puede ser:

- **Interna**, dentro del proyecto, o **externa**, con el cliente, el público, medidos de comunicación u otros proyectos.
- **Formal**, con informes, memorandos e instrucciones, o informal, por correo electrónico y conversaciones.
- **Vertical**, entre superiores y subordinados, o **horizontal**, entre miembros del mismo rango.
- **Oficial**, con boletines o informes anuales, o **no oficial** o **extraoficial**.
- **Escrita** u **oral**.
- **Verbal**, o **no verbal**, basada en inflexiones de voz y lenguaje corporal.

Cada una de estas clasificaciones es una **dimensión**.

Algunos procesos de este área son:

1. **Planificar la gestión de las comunicaciones.** Desarrollar un enfoque y un plan para las actividades de comunicación del proyecto según las necesidades de información de cada interesado o grupo, las del proyecto y los activos disponibles.

   Un indicador de la complejidad de las comunicaciones es el número de canales de comunicación posibles, el tamaño de un grafo completo cuyos nodos son los interesados.

   El **plan de gestión de las comunicaciones** suele contener los requisitos de comunicación de los interesados; la información que debe ser comunicada, incluyendo formato, contenido y nivel de detalle; el responsable de comunicar; la persona o grupo receptor; los métodos o tecnologías para transmitir; la frecuencia de la comunicación; un glosario de terminología común; un **proceso de escalamiento** con los plazos y la cadena de mando para el escalado de polémicas que no pueden resolverse en un nivel inferior; un método de actualización del plan; pautas para reuniones sobre el estado; reuniones del equipo de proyecto; reuniones electrónicas, y correo electrónico.

2. **Gestionar las comunicaciones.** Garantizar que la recopilación, creación, distribución, almacenamiento, recuperación, gestión, monitoreo y disposición final de la información del proyecto sean oportunos y adecuados.

3. **Monitorear las comunicaciones.** Asegurar que se satisfagan las necesidades de información del proyecto y de sus interesados.

7.4.2. Gestión de los interesados del proyecto

Los principales procesos son:

1. **Identificar a los interesados.** Para hacer un **análisis de interesados**:

   a) Identificar a los interesados potenciales e información relevante como su rol, departamento, intereses, conocimiento, expectativas y niveles de influencia.

   b) Analizar el impacto o apoyo potencial que pueden generar.

   c) Valorar cómo reaccionarán ante varias situaciones.

Algunos modelos de clasificación de interesados son las redes de poder–interés, de poder–influencia y de influencia–impacto, entendiendo como influencia el nivel de participación y como impacto la habilidad de efectuar cambios, así como las clases de interesados según su poder, urgencia y legitimidad.

2. **Planificar la implicación de los interesados.** En el **plan de implicación de los interesados** se pueden incluir estrategias para comprometer a los interesados; necesidades de comunicación; relaciones entre los interesados; formato y frecuencia de las comunicaciones, y técnicas de seguimiento y actualización del plan. Podemos clasificar a los interesados según su nivel de compromiso en desinformados, reticentes, neutrales, promotores e impulsores.

3. **Gestionar la implicación de los interesados.** Actividades:

   a) Involucrar a los interesados en las etapas del proyecto apropiadas.

   b) Gestionar las expectativas de los interesados mediante negociación y comunicación.

   c) Abordar las cuestiones de los interesados que se pueden convertir en problemas potenciales.

   d) Clarificar y resolver los problemas identificados.

4. **Monitorear la implicación de los interesados.**

Capítulo 8

Estimación de costes

Una **Estructura de Descomposición del Trabajo** (**EDT** o **WBS**, *Work Breakdown Structure*) es una forma de organizar elementos de trabajo de un proyecto descomponiendo tareas en subtareas relacionadas.

Una **línea base de costes** es una función que asocia a cada tiempo en la duración del proyecto el presupuesto asignado a este hasta entonces, e incluye la suma de la estimación de costes de actividad con una **reserva de contingencia** para cada actividad.

El **presupuesto de costes** del proyecto es el total de fondos autorizados para ejecutar el proyecto, y está formado por el valor final de la línea base, llamado **BAC** (*Budget At Completion*), más una **reserva de gestión**, que no se distribuye como financiación.

8.1. PMBOK

En el área de conocimiento de Gestión de los Costes del Proyecto, los principales procesos de planificación son:

1. **Planificar la gestión de costes.** Definir cómo se van a estimar, presupuestar, gestionar, monitorear y controlar los costes del proyecto.

2. **Estimar los costes.** Aproximar el coste de completar las actividades según la información disponible en un momento dado. Se debe estimar todos los recursos (trabajo, materiales, equipo, servicios, instalaciones, etc.) e ir refinando la estimación durante el proyecto.

3. **Determinar el presupuesto de costes.** Sumar los costes estimados de las actividades para establecer una línea base de costes autorizada. La reserva de gestión no forma parte de la línea base de costes pero se puede incluir en el presupuesto total.

De monitoreo y control:

4. **Controlar los Costes.** Monitorizar la situación del proyecto para actualizar su presupuestos y gestionar cambios a la línea base de costes. Acciones:

   a) Influir sobre los factores que producen cambios en la línea base.

b) Asegurarse de que los cambios solicitados sean acordados.

c) Gestionar los cambios reales a medida que se produzcan.

d) Registrar los cambios pertinentes con precisión.

e) Evitar que se incluyan cambios incorrectos, inadecuados o no aprobados en el coste o el uso de recursos.

f) Monitorizar el rendimiento del coste para detectar y entender variaciones respecto a la línea base.

g) Asegurar que los posibles sobrecostes no excedan la financiación autorizada periódica y total.

h) Mantener los sobrecostes esperados en límites aceptables.

8.2. Gestión del valor ganado

EVM (*Earned Value Management*) es una técnica para obtener **proyecciones** de coste, estimaciones basadas en lo hecho hasta el momento, e integra mediciones de alcance, coste y cronograma.

El **valor planificado** (PV, *Planned Value*) es el presupuesto autorizado para completar la actividad de la WBS, el valor actual de la línea base de costes para la actividad. El **valor ganado** (EV, *Earned Value*) es una medida del trabajo realizado en términos del presupuesto autorizado. El **coste real** (AC, *Actual Cost*) es el coste incurrido para dicho trabajo.

La **variación del cronograma** (*Schedule Variation*) mide el rendimiento del cronograma en un proyecto como $SV = EV - PV$. La **variación del coste** (*Cost Variation*) mide el rendimiento del coste de un proyecto, $CV = EV - AC$.

El **índice de rendimiento del cronograma** (*Schedule Performance Index*) mide el avance logrado en comparación con el planificado, $SPI = \frac{EV}{PV}$, y el **índice del rendimiento del coste** (*Cost Performance Index*) mide el valor del trabajo completado en comparación con el coste, $CPI = \frac{EV}{AC}$.

Una **estimación hasta la conclusión** (ETC, *Estimate To Complete*) es el coste previsto para terminar el trabajo restante de la actividad, y una **estimación a la conclusión** (EAC, *Estimate At Completion*) es el coste total previsto para la actividad, $EAC = AC + ETC$.

Algunas proyecciones comunes del EAC son:

- $EAC = BAC - CV$.
- $EAC = \frac{BAC}{CPI}$.
- $EAC = AC + \frac{BAC - EV}{CPI \cdot SPI}$.

8.3. Métricas de software

Las métricas usadas para medir el desarrollo de un software pueden ser:

- **Del producto:** dependen solo del código fuente, como el número de líneas de código o el de estructuras de datos.

- **Del proceso:** dependen del entorno de desarrollo, como el tiempo empleado.

Hay 3 tipos de proyectos informáticos: de desarrollo de soluciones; de integración de tecnologías, y de implantación y personalización de soluciones estándar. Nos centramos en el primer tipo, el único para que las métricas del producto son apropiadas.

Las métricas de software son **objetivas** si dan el mismo valor cuando la usan distintas personas y **subjetivas** si dependen del juicio de expertos, y son **primitivas** si se pueden obtener directamente y **calculadas** si se deben obtener a partir de otras métricas. También se pueden clasificar como orientadas al tamaño o a la función.

8.3.1. Métrica por líneas de código

Tiene varias definiciones, pero la idea es contar todas las instrucciones del código fuente salvo comentarios y software de utilidad como pruebas. Se usa para calcular la complejidad de la aplicación, el esfuerzo total de desarrollo, el tiempo de desarrollo y el rendimiento de programación.

8.3.2. Modelo de puntos de función

Es un modelo propuesto por A. J. Albrecht de IBM en 1979, desarrollado tras analizar 22 proyectos que pasaron de definición de requisitos a pruebas de aceptación y demos y que usaron procedimientos consistentes de gestión y definición de tareas con un seguimiento preciso del tiempo.

Mide el tamaño de la funcionalidad de un producto software desde el punto de vista del usuario, como una caja negra en la que introducir y consultar datos y recibir información.

Pasos:

1. Obtener información del sistema.
2. Identificar los componentes del sistema. Estos son:
   a) **Entradas** (**EI**, *External Inputs*): Procesos en que se introducen datos y que actualizan algún archivo interno.
   b) **Salidas** (**EO**, *External Outputs*): Procesos en que se envían datos al exterior con algún tratamiento previo.
   c) **Consultas** (**EQ**, *External Queries*): Combinación de una entrada que no produce cambios en archivos y una salida que no contiene información derivada.
   d) **Ficheros internos** (**ILF**, *Internal Logic Files*): Grupos de datos relacionados entre sí, actualizados y consultados por el sistema.
   e) **Ficheros externos** (**EIF**, *External Interface Files*): Grupos de datos que se mantienen externamente.

   Se pueden identificar desde diagramas de casos de uso UML, de contexto o **DFD** (*Data Flow Diagram*) o diagramas similares.
3. Calcular el número de elementos y la complejidad, usando tablas como las del cuadro 8.1.

EI / EQ		Atributos 1–4	5–15	> 15
Ficheros accedidos	0–1	3	3	4
	2	3	4	6
	> 2	4	6	6

EO		Atributos 1–5	6–19	> 19
Ficheros accedidos	0–1	4	4	5
	2–3	4	5	7
	> 3	5	7	7

ILF		Atributos 1–19	20–50	> 50
Entidades o registros lógicos	1	7	7	10
	2–5	7	10	15
	> 5	10	15	15

ELF		Atributos 1–19	20–50	> 50
Entidades o registros lógicos	1	5	5	7
	2–5	5	7	10
	> 5	7	10	10

Cuadro 8.1: Factores de complejidad de los componentes para calcular los puntos función.

4. Obtener los **puntos de función sin ajustar**, como suma del número de componentes de cada tipo por el factor de complejidad correspondiente.

5. Obtener los **puntos de función ajustados**, multiplicando los puntos sin calcular por un factor de ajuste dado por $0{,}65 + 0{,}01A$.

   A es la suma de una valoración de 0 a 5 dada a cada uno de 14 factores: comunicación de datos, proceso distribuido, objetivos de rendimiento, configuración de explotación compartida, tasa de transacciones, entrada de datos en línea, eficiencia con el usuario final, actualizaciones en línea, lógica de proceso interno compleja, reusabilidad del código, conversión e instalación contempladas, facilidad de operación, instalaciones múltiples y facilidad de cambios. Para cada factor hay una descripción de lo que significa cada valoración de 0 a 5.

6. Calcular el esfuerzo. Se estiman las líneas de código y las horas por punto función según el entorno y el lenguaje y con esto se calculan las líneas de código y horas.

7. Calcular la duración del proyecto. Se divide el número de horas para una sola persona entre el número de personas y el resultado se divide entre las horas dedicadas al mes.

8. Calcular el presupuesto del proyecto. Depende del esfuerzo estimado, los sueldos y otros costes de operación.

Hay al menos 6 estándares ISO basados en puntos función.

8.3.3. Puntos de caso de uso

UCP (*Use Case Point*) es una unidad de medida basada en el punto función pero adaptada a casos de uso, facilitando la estimación en empresas o proyectos que usan casos de uso.

8.4. Técnicas de estimación

Para estimar el coste del software, debemos considerar factores:

- **Técnicos:** Nivel de conocimientos, formación del personal, nuevas incorporaciones, características del proyecto, complejidad, procedimientos, estándares de funcionamiento, participación en proyectos similares.
- **Personales:** Vacaciones, ausencias, enfermedades, formación, días festivos.
- **Dependencias:** De grupos no involucrados directamente en el proyecto, como dependencias externas con otras organizaciones.
- **Contingencias:** Desplazamientos, demos, repetición de actividades, calendario de trabajo impuesto, imprevistos.

Fases de la estimación:

1. **Estimación del tamaño.** Muy compleja.
2. **Estimación del esfuerzo.** En personas-días, horas-categoría, personas-mes, etc. Según el tamaño y datos previos de la organización en proyectos similares.
3. **Estimación de la planificación.** Establecer un calendario tras conocer el esfuerzo y los recursos a emplear.
4. **Estimación del costo.** Sumar el costo de los recursos según el tiempo de uso.

   Etapa general. Se da una estimación con cierto margen y se va aumentando la precisión conforme avanza el proyecto.

Una técnica de estimación puede ser basada en el tamaño del producto, basada en la experiencia y/o específica para proyectos informáticos, y estas categorías no son excluyentes. Una teoría específica para proyectos informáticos es la teoría de la ciencia del software de Halstead.

8.4.1. Técnicas basadas en la experiencia

La técnica por **analogías** (***top-down***) genera una estimación inicial no detallada mediante ***Rough Order of Magnitude*** (**ROM**), consistente en describir un rango amplio de valores para el esfuerzo comparando el proyecto con otros previos.

Esto nos da una estimación rápida en muy poco tiempo, y es apropiado para primeras aproximaciones de proyectos que no está claro que se vayan a realizar. Se pueden considerar varias soluciones alternativas y elegir la mejor.

Los **métodos paramétricos** cuantifican datos de proyectos anteriores y los combinan con características o **parámetros** del proyecto actual para estimar los costes del proyecto con un modelo matemático más o menos complejo. Los modelos paramétricos de estimación de software suelen ser complejos, con muchas variables, y son más fiables cuando son escalables, los parámetros son fácilmente cuantificables y la información histórica es más exacta.

La **técnica Delphi** consiste en que se determina qué opiniones considerar, se crea y distribuye un cuestionario sobre el coste, se recogen y tabulan las respuestas y, si no hay consenso, se distribuyen las respuestas y se hace otra ronda de recogida de datos del cuestionario, repitiendo hasta que haya consenso.

8.4.2. Técnicas basadas en el tamaño del producto

En la **estimación tarea a tarea** (***bottom-up***), se divide el trabajo en tareas individuales, se definen las tareas y actividades y, de cada tarea, se valora aproximadamente el tamaño y la complejidad, se estima la duración y se asignan recursos.

El **modelo de distribución porcentual** es intermedio entre el *bottom-up* y los modelos paramétricos. El equipo de desarrollo estima con más detalle la fase de desarrollo y el tiempo para el resto de fases se calcula según una distribución de porcentajes basada en la experiencia de la organización.

8.4.3. Técnicas específicas a proyectos informáticos

La técnica por **factores de complejidad y tamaño** combina ideas de los métodos paramétricos con Delphi. En esta se completan dos cuestionarios, uno sobre la fase de diseño y otro sobre la de implementación, sobre distintos aspectos, y se asigna peso a las respuestas de cada participante según su experiencia.

Los puntos función y puntos por casos de uso pueden usarse para estimar ya que se pueden determinar antes de escribir el código.

COCOMO (*COnstruction COst MOdel*) es un modelo paramétrico desarrollado por Barry Boehm y publicado en 1981. **COCOMO II** es una revisión de COCOMO de 1995 que contempla infraestructura, generadores de aplicaciones, aplicaciones con componentes y sistemas integrados, además de aplicaciones desarrolladas por usuarios finales que no precisan estimación del tiempo.

Incluye 3 modelos: **composición de aplicaciones** (**ACM**, *Application Composition Model*), **diseño temprano** (**EDM**, *Early Design Model*) y **pos-arquitectura** (**PAM**, *Post-Architecture Model*). De todos los tipos de productos salvo el último, en las aplicaciones con componentes se usa ACM para estimar el tamaño en puntos objeto, mientras que para el resto se usa ACM durante el prototipado, EDM para las primeras etapas del desarrollo y PAM en la etapa de desarrollo, con estimaciones en puntos función y/o líneas de código fuente.

Capítulo 9

Planificación y seguimiento

La mayoría de proyectos informáticos empiezan con una discordancia entre las prestaciones deseadas y los recursos disponible, y se debe moldear el conjunto de recursos y/o el de prestaciones para que concuerden.

La **planificación** busca suministrar una estructura que permita al director hacer estimaciones razonables de recursos, coste y agendas mediante dos actividades principales: definir el alcance del proyecto y estimar los recursos requeridos para llevar a cabo el desarrollo.

Un **modelo de programación** es una representación del plan que incluye duración, dependencias y otras restricciones. De este se obtiene un **cronograma**, que representa actividades con fechas planificadas y recursos. Una **línea base** es una instancia del cronograma usada para planificar el proyecto.

9.1. PMBOK

El área de conocimiento Gestión del Cronograma del Proyecto incluye los procesos:

- De planificación:
    1. **Planificar la Gestión del Cronograma.** Establecer políticas, procedimientos y documentación para planificar, desarrollar, gestionar, ejecutar y controlar el cronograma.
    2. **Definir las Actividades.** Identificar las acciones específicas a realizar para elaborar los entregables.
    3. **Secuenciar las Actividades.** Identificar y documentar las relaciones entre las actividades.
    4. **Estimar la Duración de las Actividades.** Aproximar la cantidad de periodos de trabajo necesarios para terminar cada actividad con los recursos estimados.
    5. **Desarrollar el Cronograma.** Analizar la duración de las actividades, sus requisitos de recursos y las restricciones de cronograma para crear el cronograma.

- De monitoreo y control:

  6. **Controlar el cronograma.** Seguir el estado del proyecto para actualizar su avance y gestionar cambios a la línea base del cronograma. Actividades:
     a) Determinar el estado actual del cronograma.
     b) Influir sobre los factores que cambian el cronograma.
     c) Determinar que el cronograma del proyecto ha cambiado.
     d) Gestionar los cambios reales a medida que suceden.

9.2. Método de diagramación por precedencia

El **PDM** (*Precedence Diagram Method*) es una técnica para construir un modelo de programación basada en diagramas de grafo dirigido cuyos nodos son actividades y cuyas flechas son dependencias entre actividades. Una dependencia de A a B indica que hasta que no empieza o termina A no puede empezar o terminar B, lo que nos da 4 tipos de dependencias:

- **Final a inicio:** Lo más común. Hasta que no se ensambla el hardware no se puede desplegar el software.
- **Final a final:** Hasta que no termine el desarrollo no puede terminar la documentación.
- **Inicio a inicio:** Hasta que no empiece el desarrollo no pueden empezar las pruebas.
- **Inicio a final:** Hasta que no empiece a mantenerse el nuevo sistema no se puede retirar el anterior.

Normalmente una flecha se une a la parte izquierda o derecha de un nodo según si ese lado de la dependencia es de inicio o fin, respectivamente.

9.3. Desarrollo del cronograma

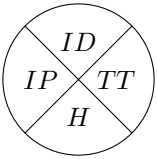
El **método de la ruta crítica** o **CPM** (*Critical Path Method*) estima la duración mínima del proyecto sin tener en cuenta las limitaciones de recursos, calculando para cada actividad:

- El **inicio más pronto** (IP) y la **terminación más pronta** (TP), las fechas más tempranas en las que el proyecto puede empezar y terminar según las dependencias. TP resulta de añadir a IP la duración de la actividad.
- El **inicio más tardío** (IT) y la **terminación más tardía** (TT), las fechas más posteriores en que el proyecto puede empezar y terminar sin retrasarlo respecto a su duración mínima.

Para la precedencia de actividades usa un diagrama **AOA** (*Activity On Arrow*), que representa los hitos de comienzo y fin de tareas como nodos las actividades como flechas, o líneas de izquierda a derecha, de un nodo origen a un nodo fin, con una relación de fin a inicio de las actividades entrantes a un nodo a las salientes.

Los ejes se etiquetan con su nombre, identificación y duración, por ejemplo con una línea encima para el nombre, otra para una letra de identificación, otra con su numeración jerárquica en el WBS entre paréntesis y una línea debajo con la duración.

Los nodos tienen el siguiente aspecto:



Hay un único nodo de inicio (sin arcos entrantes) y un único nodo de final (sin arcos salientes).

Pasos:

1. Se crea un AOA, rellenando de los nodos solo el campo ID con un identificador numérico secuencial en orden topológico del nodo de inicio al de final.

2. El campo IP es el IP de los arcos salientes, cuyo TP es la suma del IP más su duración. En orden, el IP del primer nodo se rellena con 0 y el del resto con el máximo TP de los arcos entrantes.

3. El campo TT es el TT de los arcos entrantes, cuyo IT resulta de restar al TT su duración. En orden inverso, el TT del primer nodo se rellena con su IP y el del resto con el mínimo IT de los arcos salientes.

4. El campo H de cada nodo se rellena con su **holgura**, $H = TT - IP$.

Entonces el **camino crítico** está formado por los hitos con holgura 0 y las tareas entre ellos, que son las tareas que no pueden retrasarse para que el proyecto no se demore.

A veces se usan **tareas ficticias** o ***dummy***, tareas de duración 0 sin nombre ni identificación que se añaden para establecer precedencias en el orden de otras cuando estas no se pueden establecer de otra forma, y que pueden variar las fechas y el camino crítico.

Otras formas de representar el cronograma:

- **Diagrama de barras** o **de Gantt.** De los más usados. Proporcionan información sobre el calendario.

- **Diagrama de hitos,** como uno de Gantt pero que solo identifican el inicio o el fin programado de los entregables más importantes y las interfaces externas clave.

- **PERT** (*Program/Project Evaluation and Review Technique*), similar a CPM pero estimando la duración de una tarea según una ponderación estadística entre 3 valores: pesimista, realista y optimista.

PERT o CPM se usan cuando el proyecto es muy crítico, de alto riesgo o incertidumbre, con participación de muchas personas u organizaciones, técnicamente complejo o con actividades dispersas geográficamente. Si tiene menos de 20 actividades se aconseja usar técnicas más sencillas como los diagramas de Gantt o de hitos, y si tiene más de 100 sucesos hay que usar una herramienta de gestión de proyectos.

9.4. Evaluación económica

El coste para el cliente es la suma del precio que se ofrece al cliente y el presupuesto de lo que aporta y lo que contrata directamente el cliente según su valoración.

La preparación del presupuesto de costes consiste en sumar los costes estimados de actividades individuales o paquetes de trabajo para establecer una línea base de coste. Incluye dos tipos de costes:

- **Costes directos,** generados para beneficio exclusivo del proyecto. Salarios, software adquirido específicamente, etc.
- **Costes indirectos,** generales de la organización pero asignados en parte al proyecto. Electricidad, teléfono, personal administrativo, alta dirección que dedica parte del tiempo al proyecto, etc.

Un posible desglose puede ser:

1. Software
   - 1.1. Software de base
   - 1.2. Sistemas de gestión de bases de datos
   - 1.3. Software de comunicaciones
   - 1.4. Software de desarrollo
   - 1.5. Software de aplicaciones
     - 1.5.1. Estándar
     - 1.5.2. Adaptado
     - 1.5.3. A medida
   - 1.6. Mantenimiento de software existente
2. Hardware
   - 2.1. ...
3. Comunicaciones
   - 3.1. ...
4. Personal
5. ...

9.5. Memoria del proyecto

Organizamos los documentos de lo general a lo particular. Como estos sirven de base para realizar el sistema, la ordenación interna de cada uno debe amoldarse a la señalada en la planificación temporal de trabajos. Cada documento debe ser completo en sí mismo, y se divide en tantas partes o capítulos como contratistas, suministradores, empresas de servicio, unidades de servicios, departamentos, etc. se prevea que han de ejecutar los trabajos. Estas partes deben tener sentido en sí mismas y en relación con el conjunto.

Una posible propuesta para los capítulos de un proyecto:

1. Introducción. Objeto, antecedentes y bases.
2. El sistema de información en la situación actual. Estado de su informatización.
3. Estudio y selección de alternativas. Viabilidad del proyecto.
4. Ingeniería de la alternativa seleccionada. Uno o varios capítulos de estudio técnico.
5. Ingeniería de calidad.
6. Programación temporal de la ejecución del proyecto y de la puesta en servicio del sistema.
7. Normas para la explotación del nuevo sistema.
8. Presupuesto y estudio económico y financiero.

   Anejos a la memoria.

9.6. Seguimiento y control

Un proceso está **bajo control** si se están alcanzando los hitos a tiempo y con los recursos y la calidad estimados y sigue siendo económicamente aceptable, y está **fuera de control** en otro caso. El objetivo del seguimiento es determinar si el proyecto está o no bajo control.

PMBOK define los siguientes procesos en el grupo de monitoreo y control, que tratan de asegurar que el trabajo real va de acuerdo al plan, obteniendo información del avance del proyecto, comparándola con metas y planes, revelando cuándo y dónde existen desviaciones y poniendo en marcha acciones correctoras.

- Monitorear y controlar el trabajo del proyecto.
- Realizar el **control integrado de cambios**. Se hace desde el inicio del proyecto hasta su conclusión para gestionar las solicitudes de cambio una vez se han definido las líneas base. Pasos:

  1. Revisar todas las solicitudes de cambio.
  2. Aprobar y gestionar cambios a entregables, documentos del proyecto y al plan para la dirección del proyecto.
  3. Comunicar las decisiones.

  Las solicitudes pueden incluir **acciones correctivas** para re-alinear el desempeño de trabajo con el plan y **acciones preventivas** para que el desempeño futuro se esté alineado con el plan.
- Validar el alcance.
- Controlar el alcance.
- Controlar el cronograma.

- Controlar los costos.
- Controlar la calidad.
- Controlar los recursos.
- Monitorear las comunicaciones.
- Monitorear los riesgos.
- Controlar las adquisiciones.
- Monitorear la implicación de los interesados.

Las principales acciones a realizar para gestionar una crisis en un proyecto son:

1. **Anuncio y publicidad general del problema.** Se comunica a todos el problema para que interioricen la situación y ayuden sin reservas, pues si se enteran de este informalmente pensarán que está controlado.
2. **Re-asignación de responsabilidades y autoridad.** Se reasignan recursos, paralizando algunas tareas y asignando sus empleados y recursos a otras, cuidando de aclarar las responsabilidades y definir quién puede tomar decisiones sobre qué.
3. **Actualizar la información de situación.** Planificar reuniones para que los implicados alineen trabajos hacia la solución, trabajando en grupo para obtener soluciones más creativas, y dejar constancia de lo intentado para evitar volver a cometer el mismo error.
4. **Relajar restricciones sobre recursos.** Los recursos suelen ser limitados por estar compartidos con otros proyectos, así que se hacen excepciones permitiendo el uso de equipos más rápidos, aumentando el soporte administrativo, etc.
5. **Poner al personal a trabajar al máximo.** Que los trabajadores trabajen tanto como puedan, que tengan los móviles siempre encendidos para facilitar la comunicación, suprimir reuniones de departamento, aplazar cursillos, etc.[1]
6. **Establecer la fecha de finalización de la crisis.** Marcar un plazo razonable, de no más de 30 días, para que termine la crisis, y replantearse la viabilidad del proyecto si se sobrepasa el límite ya que la gente no puede vivir con tanto estrés tanto tiempo.

En la recuperación tras la crisis:

1. **Eliminar al personal innecesario.**[2]
2. **Re-planificar el proyecto tras la crisis.** Determinar cómo esta ha afectado al cronograma, el coste, el alcance, etc.
3. **Realizar un estudio *post mortem* de la crisis.** Examinar qué fue mal, identificar problemas sistemáticos que podrían evitarse y documentar lo aprendido.

[1] Comprar y formar a esclavos del África subsahariana, añadir carteles de «El gran hermano te vigila», etc.
[2] No en el sentido de matarlos sino en el de quitarles su medio de vida.

9.7. Cierre del proyecto

En PMBOK, el principal proceso del grupo de procesos de cierre es «Cerrar el proyecto o fase», que termina todas las actividades del proyecto, fase o contrato y aborda también el cierre anticipado del proyecto. Actividades:

- Medir la satisfacción de los interesados.
- Confirmar la entrega y aceptación formal del entregables por el cliente.
- Transferir resultado a la siguiente fase, producción u operaciones.
- Liberar recursos.[3]
- Actualizar el registro de lecciones aprendidas, analizando y documentando las razones si un proyecto se da por terminado antes de completarse.

[3] En C, esto se hace con `free(recurso)`.

Capítulo 10

Análisis y gestión de riesgos

Un **riesgo de un proyecto** es un evento o condición incierto, llamado **evento de riesgo**, con una o más **causas** de que pueda ocurrir, y que de producirse tiene un efecto positivo o negativo (el **impacto**) en uno o varios objetivos del proyecto. Si el efecto es positivo, el riesgo es **positivo** o una **oportunidad**, y si es negativo, el riesgo es **negativo** o una **amenaza**.

La **gestión de riesgos del proyecto** es una serie de procesos para identificar; analizar, y planificar e implementar respuestas a, los riesgos del proyecto, para aumentar la probabilidad y el impacto de los riesgos positivos y disminuir los de los negativos.

Distinguimos entre **riesgos conocidos**, identificados y analizados para los que se puede planificar una respuesta usando la reserva de contingencia, y los **riesgos desconocidos**, que no pueden gestionarse proactivamente y a los que se responde con la reserva de gestión.

La **mitigación de riesgos** consiste en planificar con antelación para reducir el impacto o la probabilidad de los riesgos. Una **salvaguarda** es un procedimiento o mecanismo tecnológico que reduce el impacto o la probabilidad de un riesgo.

El área de conocimiento Gestión de los Riesgos del Proyecto de PMBOK tiene los siguientes procesos:

- De planificación:
    1. Planificar la gestión de los riesgos.
    2. Identificar los riesgos.
    3. Realizar el análisis cualitativo de riesgos.
    4. Realizar el análisis cuantitativo de riesgos.
    5. Planificar la respuesta a los riesgos.
- De ejecución:
    6. Implementar la respuesta a los riesgos.
- De monitoreo y control:
    7. Monitorear los riesgos.

Escala	Probabilidad	Tiempo	Coste	Calidad/alcance
Muy alto	> 70 %	> 6 meses	> 5 M$	Impacto muy significativo general.
Alto	51–70 %	3–6 meses	1–5 M$	Impacto significativo general.
Mediano	31–50 %	1–3 meses	0.5–1 M$	Algún impacto en áreas clave.
Bajo	11–30 %	1–4 semanas	100–500 K$	Impacto menor general.
Muy bajo	1–10 %	1 semana	< 100 K$	Impacto menor en funciones secundarias.
Nulo	< 1 %	Sin cambio	Sin cambio	Ningún cambio.

Cuadro 10.1: Ejemplo de escalas para una empresa grande.

10.1. Planificación de la gestión de riesgos

Una **estructura de desglose de los riesgos** (**RBS**, *Risk Breakdown Structure*) es una estructura de clasificación jerárquica de las fuentes de riesgos para identificar y categorizar riesgos.

La probabilidad de un riesgo se puede medir con una **escala relativa** o **cualitativa**, con valores como «muy baja», «baja», «mediana», «alta» o «muy alta», o con **probabilidades numéricas** aproximadas, con una escala general cuantitativa y discreta con valores como 0,1, 0,3, 0,5, 0,7 y 0,9. De igual modo, el impacto en cada uno de los objetivos «clásicos» ((coste, tiempo, alcance y calidad) se puede medir con una escala cualitativa, con valores como «muy bajo», «bajo», «moderado», «alto» y «muy alto», o con una numérica, con valores como 0,05, 0,1, 0,2, 0,4 y 0,8. Una escala cualitativa aparece en el cuadro 10.1.

Una **matriz de probabilidad e impacto** es una tabla de la forma:

Probabilidad	**Amenazas**			**Oportunidades**			
p_1	p_1c_1	$\cdots$	p_1c_m	p_1c_m	$\cdots$	p_1c_1	p_1
$\vdots$	$\vdots$					$\vdots$	$\vdots$
p_n	p_nc_1	$\cdots$	p_nc_m	p_nc_m	$\cdots$	p_nc_1	p_n
	c_1	$\cdots$	c_m	c_m	$\cdots$	c_n	
	Impacto negativo			**Impacto positivo**			

Donde $1 \geq p_1 \geq \ldots \geq p_n \geq 0$ es la escala de probabilidades y $0 \leq c_1 \leq \ldots \leq c_n \leq 1$ es la escala de impactos. La **exposición** a un riesgo es la probabilidad de ocurrencia multiplicada por el impacto.

Un **plan de gestión de riesgos** describe como se estructurará y realizará la gestión de riesgos en el proyecto, e incluye:

- **Metodología:** Métodos, herramientas y fuentes de información que se pueden usar para gestionar los riesgos.
- **Roles y responsabilidades:** Líder, apoyo y miembros del equipo de gestión de riesgos para cada tipo de actividad del plan. Personas asignadas a esos roles y sus responsabilidades.
- **Presupuesto:** Recursos y costes estimados necesarios para la gestión de riesgos, para incluirlos en la línea base de coste.

- **Calendario:** Cuándo y con qué frecuencia se realizará el proceso de gestión de riesgos en el ciclo de vida del proyecto y qué actividades de gestión de riesgos se incluirán en el cronograma.
- **Categorías de riesgo,** por ejemplo con un RBS.
- **Definiciones de probabilidad e impacto de los riesgos.** Escalas a usar.
- **Matriz de probabilidad e impacto.**
- **Tolerancias revisadas de los interesados** a los riesgos. Pueden depender del proyecto.
- **Formatos de informe.** Contenido y formato del registro de riesgos y de cualquier otro informe de riesgos que se requiera. Forma de documentar, analizar y comunicar los resultados de los procesos de gestión de riesgos.
- **Seguimiento.** Cómo se registrarán todas las facetas de las actividades de riesgo para beneficio del proyecto actual, futuras necesidades y lecciones aprendidas, y si serán auditados los procesos de gestión de riesgos y cómo.

10.2. Identificación de riesgos

Se puede hacer con **tormenta de ideas**, partiendo de un RBS; mediante **entrevistas** a participantes experimentados, interesados y expertos en la materia, o con **listas de verificación**, con elementos a ser considerados como información histórica o genérica, y asegurándose de explorar elementos no presentes en la lista.

Los participantes en el proceso pueden ser el director del proyecto, los miembros del equipo, especialistas en gestión de riesgos, etc., aunque se fomenta que participen todos los *stakeholders*.

Podemos analizar los datos con **análisis de causa raíz**, partiendo de una amenaza u oportunidad y describiendo causas subyacentes; **análisis de supuestos y restricciones**, explorando la validez de las hipótesis establecidas al concebir los planes y considerando el riesgo de que estas no se cumplan; con la técnica Delphi, o con **análisis DAFO**, examinando las debilidades, amenazas, fortalezas y oportunidades.

Para el análisis de causa raíz se puede usar un **diagrama de causa y efecto**, **de Ishikawa** o **de espina de pescado**, en el que aparece una línea horizontal con un cuadro con el nombre del riesgo a la derecha y de la que parten líneas hacia arriba y abajo algo inclinadas hacia atrás que terminan en un cuadro con el nombre de una categoría de causas (personas, proceso, equipos, material, entorno, gestión) y de las que parten, a la derecha, segmentos horizontales de línea etiquetados con una causa.

Se puede usar un **diagrama de flujo**, cuyos elementos son cuadros que indican actividades y, salvo el último, tienen una flecha hacia el siguiente elemento a ejecutar, y rombos que indican preguntas o actividades de aprobación y de los que parten dos flechas etiquetadas con «sí» y «no» que apuntan al siguiente proceso según si lo indicado se aprueba o no.

Un **diagrama de influencias** o **causal** está formado por cuadros con condiciones, eventos o magnitudes («preferencia por tal», «coste/ventas de tal», etc.) y flechas etiquetadas con «+» o «-» indicando que una condición influye positiva o negativamente en otra.

Los riesgos mas comunes en desarrollo de software son:

1. Cambio de requisitos.
2. Meticulosidad en requisitos o desarrolladores.
3. Escatimar en calidad.
4. Planificación demasiado optimista.
5. Diseño inadecuado.
6. Síndrome de la panacea.
7. Desarrollo orientado a la investigación.
8. Personal mediocre.
9. Error en la contratación.
10. Diferencias entre el personal de desarrollo y los clientes.

10.3. Análisis de riesgos

El **análisis cualitativo** consiste en priorizar riesgos según características como su probabilidad de ocurrencia e impacto para concentrar los esfuerzos en los riesgos de alta prioridad.

Algunas técnicas son:

- Evaluación de probabilidad e impacto de los riesgos, según las definiciones proporcionadas en el plan de gestión de riesgos.
- Uso de una matriz de probabilidad e impacto.
- Evaluación de la calidad de datos sobre riesgos, examinando el grado de entendimiento del riesgo y la exactitud, calidad, fiabilidad e integridad de los datos.
- Categorización de riesgos, por fuente de riesgo en el RBS, por área de proyecto en el WBS, etc.
- Evaluación de su urgencia, según indicadores como el tiempo para dar una respuesta, los síntomas, las señales de advertencia, la calificación y si se requiere respuesta a corto plazo.

El análisis **cuantitativo** analiza el efecto de los riesgos y los califica numéricamente. Incluye cuantificar los posibles resultados del proyecto y sus probabilidades.; evaluar la probabilidad de lograr objetivos específicos del proyecto; identificar los riesgos que requieren más atención cuantificando su contribución relativa al riesgo general, e identificar objetivos de coste, cronograma y alcance realistas y viables dados los riesgos del proyecto.

Algunas técnicas son:

- **Entrevistas** a los interesados relevantes para ayudar a determinar estimaciones de probabilidad e impacto baja, más probable y alta (caso peor, medio y mejor).

- **Distribución de probabilidades** creada a partir de esos tres casos con distribución beta o triangular, con densidad de probabilidad no nula solo entre los casos mejor y peor y máxima en el caso más probable.
- **Árbol de decisiones,** con 4 niveles de izquierda a derecha:
    1. Nodo raíz, definición de la decisión que hay que tomar. Se une a la derecha con un cuadrado sólido del que parten los hijos.
    2. Nodos de decisión, cada uno con una posibilidad a tomar y su coste. Se una a la derecha con un círculo sólido del que parten los hijos.
    3. Nodos de oportunidad, con un escenario y el beneficio si este ocurre y con el eje que lo une al padre etiquetado con la probabilidad.
    4. Valor neto de ruta. Uno por nodo de oportunidad, representado con un triángulo sólido con una de las puntas a la izquierda y etiquetado con el beneficio de la oportunidad menos el coste de la decisión.

  Se etiqueta cada nodo de decisión con el valor neto de ruta medio ponderado por probabilidad, y el nodo raíz con el mayor de estos valores medios.

10.4. Planificación de respuesta

Consiste en seleccionar estrategias para abordar la exposición al riesgo y tratar riesgos individuales reduciendo las amenazas y maximizando las oportunidades.

Las estrategias pueden ser:

1. **Escalar.** Gestionar a nivel de programa o portafolio.
2. **Evitar** la amenaza o proteger del impacto, o **explorar** la oportunidad y asegurarse de que ocurra. Para riesgos de alta prioridad.
3. **Transferir** la amenaza cediendo la titularidad a cambio de una **prima de riesgo**, o **compartir** la oportunidad transfiriendo la propiedad a un tercero más capacitado por una prima de riesgo.
4. **Mitigar** la amenaza reduciendo su exposición o **mejorar** la oportunidad aumentando su exposición.
5. **Aceptar** el riesgo sin tomar medidas proactivas. Para riesgos de baja prioridad o sin respuesta. Puede prepararse una respuesta **activa**, con contingencias, o una **pasiva**, monitoreando el riesgo.

La respuesta puede ser **activa**, con contingencias, o **pasiva**, revisando periódicamente el riesgo.

10.5. Control de riesgos

Consiste en identificar, analizar y planificar nuevos riesgos; realizar el seguimiento de los riesgos identificados y los de la lista de supervisión; re-analizar los riesgos existentes; seguir las condiciones que disparan los planes para contingencias; seguir los riesgos residuales, y revisar la ejecución de las respuestas a los riesgos mientras se evalúa su efectividad.

Capítulo 11

Métodos ágiles

Son útiles para software de gestión o de tamaño pequeño o medio con requisitos que cambian rápidamente. No lo son en desarrollo de software a gran escala con equipos de desarrollo en sitios distintos e interacciones complejas con otros sistemas hardware y software, ni en sistemas críticos en los que hace falta un análisis detallado de los requisitos para comprender las implicaciones de seguridad informática y humana.

Cuando se aplica un método pesado a una aplicación pequeña o mediana, la sobrecarga de trabajo domina el desarrollo, dedicando más tiempo a cómo se desarrollará el sistema que a la programación y la prueba, y re-diseñando y re-documentando cada vez que se cambia un requisito, por lo que a finales de los 90 surgen métodos ágiles, iterativos e incrementales, que atienden más al software que al diseño y la documentación.

Según el **manifiesto ágil**, las principales características de los métodos ágiles son la implicación del cliente en el desarrollo; la entrega incremental, en la que el cliente especifica los requisitos a incluir en cada incremento; el enfoque en las personas del equipo para sacar partido a las destrezas de cada una permitiéndoles usar su forma de trabajo; asumir el cambio, diseñando el sistema para acomodar los cambios, y mantener la simplicidad, tanto en el software como en el proceso de desarrollo, eliminando activamente la complejidad donde sea posible.

La documentación se reduce al mínimo y no necesita ser perfecta, y se reduce la especificación de requisitos.

Los métodos ágiles mejoran la sostenibilidad medioambiental ya que, como la reutilización, mejoran la calidad y la productividad, y además hay menos documentación, más adaptación a los cambios, mejor comunicación con el equipo, menos trabajo, más satisfacción de desarrolladores y clientes, y todo eso es bueno para el medio ambiente porque sí.

Algunos métodos ágiles son XP, Crystal, FDD y el Modelado Ágil de Ambler.

11.1. Historias de usuario

Son descripciones en jerga de usuario de objetos de alto nivel de la organización o el cliente, normalmente formadas por 3 frases escritas por el usuario («como *rol*, quiero que el sistema haga *cierta funcionalidad* para obtener *cierto beneficio*»), y tienen pruebas de aceptación asociadas.

Se descomponen en **tareas** a realizar para implementar la historia y se agrupan en **temas**, categorías de historias de usuario dentro del proyecto con una ***epic*** asociada, una historia de

usuario general de alto nivel, sin pruebas de aceptación.

Son la forma normal de especificar requisitos en los métodos ágiles, y suelen describir requisitos funcionales. Los no funcionales se pueden escribir como tareas si afectan claramente a alguna historia o se pueden reconvertir a historias de usuario.

Las buenas historias de usuario son independientes, negociables, valiosas, estimables, pequeñas y verificables (**INVEST**: *Independent, Negotiable, Valuable, Estimable, Small & Testable*).

Una **hoja de ruta** (***roadmap***) es un plan de alto nivel que describe cómo va a ir evolucionando el producto, con al menos versiones del producto, fechas previstas de lanzamiento y funcionalidad.

11.2. SCRUM

Es un modelo de referencia que define un conjunto de prácticas y roles y puede tomarse como punto de partida para definir el proceso de desarrollo. En inglés significa «melé», una táctica de rugby en la que los miembros del equipo se apelotonan.

El marco de trabajo lo forman:

- **Equipos Scrum** multifuncionales, autogestionados y responsables de todo, formados por:
  - Un *product owner*, cliente o representante del mismo que define el valor del producto desde el punto de vista del negocio.
  - Un *scrum master*, que asegura un correcto seguimiento de Scrum eliminando obstáculos y protegiendo al equipo. Es interlocutor con entidades externas, pero no es el líder del equipo.
  - Desarrolladores, responsables de entregar incrementos de producto terminados. Lo ideal es que sean entre 3 y 9.
- **Eventos** del proceso.
- **Artefactos.**
  - ***Product backlog*:** Relación dinámica de requisitos, no demasiado detallados al principio, con descripción, priorización, tamaño y valor. Lo gestiona el *product owner* según las necesidades planteadas por clientes y usuarios, pudiendo agregar requisitos en cualquier momento para acomodar cambios.

    Puede contener historias de usuario, casos de uso, descomposiciones de requisitos, incidencias detectadas, posibles mejoras, realimentación procedente de un incremento, etc.
- **Reglas** que relacionan los eventos, roles y artefactos y gobiernan la interacción entre ellos.

Fases:

1. **Planificación** de la versión del software a construir.
2. **Fase iterativa**, con varios *sprints* o iteraciones.

3. **Cierre:** preparación de la versión del software que se va a instalar y la documentación final.

Un *sprint* es un bloque de tiempo de un mes o menos en el que se crea un incremento de producto terminado, utilizable y potencialmente desplegable. Es de tamaño fijo, aunque se puede acabar antes si se logran los objetivos, que se establecen al principio el mismo y no se cambian a mitad. Aunque no es lo normal, se puede cancelar un *sprint* si los objetivos quedan obsoletos. Cada *sprint* comienza inmediatamente después del anterior y contiene:

- Planificación (*sprint planning*). Reunión de todo el equipo al inicio del *sprint*, de una jornada (8 horas) para un *sprint* de un mes. Con el *product backlog*, se establece qué puede entregarse en el incremento y cómo se conseguirá hacer el trabajo para implementarlo, y se crea un *sprint backlog*.

- Reuniones diarias (*daily scrum*). Los desarrolladores y el *scrum master* se reúnen durante no más de 15 minutos para actualizar la lista del *sprint*. Cada miembro explica qué hizo el día anterior para ayudar a lograr el objetivo, qué hará ese día y si ve algún impedimento para lograr el objetivo.

- Trabajo de desarrollo.

- Revisión (*sprint review*). Reunión de todo el equipo al final del *sprint*, de media jornada para un *sprint* de un mes, en la que se presenta el incremento, se plantean sugerencias, se revisan los problemas que aparecieron y cómo se resolvieron y se establece qué ha de cambiar para dar más valor al producto. Está orientada al producto y al cliente.

- Retrospectiva (*sprint retrospective*). Reunión de todo el equipo al final, de unas 3 horas para un *sprint* de un mes, donde se revisa cómo fue el último *sprint*, se identifican y ordenan los elementos más importantes que salieron bien y las posibles mejoras y se crea un plan para implementar las mejoras. Está orientada al proceso y al equipo.

La prioridad de los elementos del *product backlog* refleja el valor[1] que tiene para el negocio, o el riesgo del elemento para atacarlo de forma temprana. La **técnica MoSCoW** establece 4 niveles de prioridad: *MOst vital/must have*, *Should have*, *COuld have* y *Won't have*.

La estimación de tamaño de los elementos

[1] No necesariamente económico.