Gestión
en la ingeniería del software

Daniel Rodríguez (daniel.rodriguezg@uah.es)

Índice

1. Introducción a la gestión de proyectos

Existen múltiples definiciones de la gestión de proyectos. Destacamos la Guía del Conocimiento de Gestión de Proyectos (PMBOK – *Project Management Body of Knowledge*):

Definición
La **Gestión de proyectos** es la aplicación del conocimiento, habilidades, herramientas y técnicas a las actividades de un proyecto para cumplir los requisitos del proyecto.

Otra definición es la del glosario IEEE 610.12-90:

Definición
La **gestión en la ingeniería del software** se puede definir como la aplicación de las actividades de gestión –planificación, coordinación, medición, monitorización, control y realización de informes- para asegurar que el desarrollo y el mantenimiento del software se realiza de una forma sistemática, disciplinada y cuantificable.

En el estándar IEEE/EIA 12207, la gestión de proyectos se lleva a cabo mediante los siguientes procesos (y de forma muy similar en la guía SWEBOK y en el PMBOK):

1. **Iniciación y alcance del proyecto**. Incluye todas las actividades necesarias para decidir si el proyecto debe llevarse a cabo. Estas actividades incluyen: (i) determinación y alcance de los objetivos, (ii) estudios de viabilidad del proyecto y (iii) revisión e inspección de requisitos.

2. **Planificación del proyecto**. Incluyen las actividades de preparación del proyecto: (i) la *planificación del proceso*, donde se decide el ciclo de vida y las herramientas a utilizar; (ii) la *determinación de entregables*, (iii) la *estimación de coste, plazos y esfuerzo*, (iv) *asignación de recursos* a actividades, la planificación de la *gestión de riesgos* y la *gestión de la calidad*.

3. **Seguimiento del proyecto**. Una vez que el proyecto se esta llevando a cabo, es necesario el constante control del mismo para realizar acciones correctivas cuando la planificación y el actual curso del proyecto divergen. Además se debe mantener la moral del equipo y dependiendo del proyecto, mantener a la dirección de la organización informada del progreso.

4. **Revisión y evaluación del proyecto**. En esta actividad se analiza si los requisitos han sido satisfechos y se evalua la eficiencia y eficacia con la que se ha llevado a cabo el proyecto.

5. **Cierre del proyecto**. Análisis *post-mortem* del proyecto, donde se aprende de los errores y se analizar posibles mejoras en futuros proyectos.

Para ello, los gestores de proyectos, que deben mantener un balance entre los tres parámetros principales que definen un proyectos software a lo largo de su ciclo de vida:

- La funcionalidad que ha de proporcionar
- El plazo en que se debe desarrollar
- Los recursos de los que disponemos para ello (personal, dinero, herramientas, métodos, etc.)

Estos parámetros se suelen ver como independientes, pero realmente influyen unos en otros. Por ejemplo, si necesitamos aumentar la funcionalidad, también necesitaremos probablemente aumentar el tiempo y/o los recursos. Por otro lado si reducimos el plazo, o bien reducimos la funcionalidad o la calidad del producto a desarrollar. Si aumentamos el personal, se pueden reducir los plazos pero disminuye la productividad al incrementar la comunicación entre el personal, etc. En resumen, necesitamos una solución de compromiso entre las tres dimensiones que componen lo que se ha dado en llamar el *"triángulo mágico"* (ver figura 1). Los gestores de proyectos se encargan de

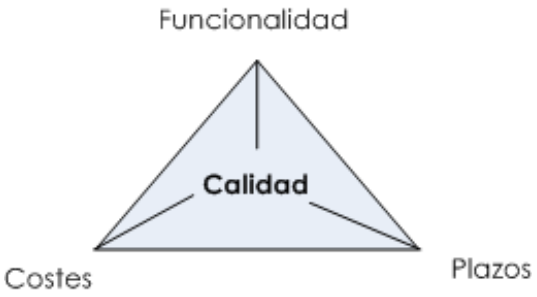

Figura 1: El triángulo mágico en la gestión de proyectos

equilibrar estas variables a lo largo del proyecto para mantener los objetivos del mismo.

Es evidente que para lleva a cabo la gestión de proyectos, la organización debe de establecer los **procesos de medición** y como dijo T. DeMarco []: "*No puedes controlar lo que no puedes medir*".

2. La estimación de coste, plazos y esfuerzo

Una estimación en general se puede definir como el proceso de determinar una variable desconocida a partir de otras conocidas. La estimación de coste, plazos y esfuerzo es parte esencial de la planificación de cualquier tipo de proyectos, y evidentemente no lo es menos en el caso de la Ingeniería del Software. Ésta no es tarea fácil por múltiples razones, desde aspectos relacionados con el proyecto (por ejemplo, proyectos nuevos sin parecido a los anteriores o donde se ha de utilizar nueva tecnología, requisitos cambiantes o pobremente definidos, falta de análisis del problema, etc.) hasta otros otros relacionados con la estimación en sí misma (por ejemplo, falta de experiencia en técnicas de estimación, la utilización de métodos de estimación que no son apropiados, la falta de históricos de proyectos para reusar la experiencia, etc.).

La correcta estimación del coste y esfuerzo, es una tarea crítica ya que **subestimar** puede suponer que el proyecto se llevará a cabo con pérdidas

para la organización. Además al no poder dedicarle los recursos necesarios, el muy posible que el proyecto se retrase, que el producto resultante sea de menor calidad que la requerida, que el personal asignado tenga que hacer horas extra (repercutiendo en la moral del equipo) o que incluso al final el proyecto tenga que ser cancelado.

En el caso contrario de **sobreestimar**, se produce el efecto de lo que se conoce humorísticamente como la "ley de Parkinson": *el trabajo se expande hasta rellenar todo el volumen disponible.* Además, en el caso de licitaciones hay más posibilidades de perderlas en favor de la competencia (en licitaciones, suele utilizarse la técnica de çoste ganador-*price to win* en inglés–, donde lo que se estima es el presupuesto que gane a los competidores y después se ajustan los parámetros para llevar a cabo el proyecto en el presupuesto).

Para evitar los problemas mencionados y establecer una correcta estimación de coste y esfuerzo, necesitamos centrar los objetivos de proyecto, procurar que los requisitos estén bien especificados y recabar toda la información disponible en ese momento. Además ésta es una actividad que debería realizarse varias veces durante el ciclo de vida, ya que según avanza el proyecto, tendremos más información y si las nuevas estimaciones varían mucho de las originales, podrán tomarse las medidas correctivas necesarias.

Existen diversos métodos para realizar estimaciones, la mayoría utilizan datos de proyectos anteriores para calcular las nuevas predicciones con diferente grado de acierto. Sin embargo, no es extraño usar más de un método de estimación, ya que no hay método mejor que otro y unos pueden complementar a otros como forma de verificar las estimaciones. A continuación se describen brevemente algunas técnicas pertenecientes a cada una de las categorías en las que se pueden agrupar las técnicas de estimación:

- Estimación basada en el juicio (experiencia) de expertos. Esta técnica se basa simplemente en la experiencia y conocimiento personal.
- Modelos algorítmicos o paramétricos. El coste y esfuerzo es calculado mediante ecuaciones que tienen en cuenta distintas variables del proyecto. Los puntos de función podrían describirse en esta categoría pero por

su importancia, al ser, quizás el método mas utilizado se describen en una un sección en sí misma.

- Otros métodos. Principalmente basados en técnicas de inteligencia artificial y minería de datos.

2.1. Estimación mediante juicio de expertos

La estimación mediante juicio de expertos esta basada en el conocimiento y experiencia de proyectos similares. Aunque existen estudios empíricos que muestran que las estimaciones suelen ser bastante acertadas en sí mismas, suele utilizarse como confirmación de los resultados de otras técnicas de estimación o como factor de corrección. Este técnica de estimación tiene el inconveniente de que son obtenidas como *caja negra*, ya que no definen como han llegado a la estimación.

Entre las técnicas más conocidas destacan el método *Delphi* y *la estimación por analogía* cuando se dispone de un histórico de proyectos (una base de datos con información de proyectos pasados).

2.1.1. Método Delphi

Este método fue creado sobre 1950 en el ámbito militar donde una serie de cuestionarios y opiniones de expertos se entremezclaban para obtener predicciones.

En la ingeniería del software, B. Boehm [4] formalizó esta técnica llamada *Wideband Delphi* para las estimaciones de tamaño del software. En esta técnica un coordinador proporciona a cada experto una especificación y un formulario. El coordinador reune a los expertos para intercambiar opiniones sobre la estimación y al final de la reunión cada experto rellena su formulario de forma anónima. A continuación, el coordinador proporciona a los expertos un resumen de la estimación y se organizan nuevas reuniones para debatir la estimaciones de la ronda anterior hasta el punto donde ya no sean necesarias más revisiones.

Existen variantes de estos pasos básicos como por ejemplo, comentar las estimaciones por adelantado y justificar los costes entre los expertos.

2.1.2. Estimación por analogía de expertos

Esta técnica consiste simplemente en que la estimación del proyecto actual se realiza comparándolo con proyectos de características y ámbito similares. Obviamente, se presupone que la organización mantiene una base de datos con proyectos realizados, o utiliza algun repositorio disponible.

2.2. Puntos de función

Los puntos de función originalmente creados por Albretch sobre 1979 en IBM son una de las técnicas más usadas para la estimación de tamaño en proyectos software. Existen variantes a este modelo, siendo los más conocidos los puntos de función IFPUG y COSMIC.

De modo simplificado para entender el funcionamiento de los puntos de función en general, los puntos de función se basan en el conteo de elementos para medir su funcionalidad. Por ejemplo, en IFPUG lo que se llama puntos de función no ajustados (UPF -*Unadjusted Function Points*-) se consideran:

- *Entradas externas* se refiere a entradas de usuario como pueden ser las selecciones de menú.
- *Salidas externas* proporcionan información al usuario (por ejemplo mensajes o informes).
- *Consultas externas* son entradas interactivas que requieren respuesta por parte del sistema.
- *Ficheros externos* son interfaces con otros sistemas.
- *Ficheros internos* son ficheros (hoy día, se consideran todo tipo de entidades persistentes) manejados por el sistema.

Cuadro 1: Factor de ponderación según complejidad

	Simple	Media	Compleja
Entrada externa	3	4	6
Salida externa	4	5	7
Consultas usuario	3	4	6
Ficheros externos	7	10	15
Ficheros internos	5	7	10

Cuadro 2: Ejemplo PF no ajustados

	Simple		*Media*		*Compleja*		*Suma*
Entrada externa	2×3	+	4×4	+	1×6	=	28
Salida externa	0×4	+	6×2	+	2×7	=	26
Consultas usuario	0×3	+	8×4	+	0×6	=	32
Ficheros externos	0×7	+	2×10	+	0×15	=	20
Ficheros internos	0×5	+	2×7	+	0×10	=	14
					Total PF no ajustados:		**120**

Estos elementos clasificados según su complejidad (simple, media o compleja) por un valor de ajuste acorde a ciertas tablas de ponderación (ver tabla 2.2). Existen guías para clasificar cada elemento dentro de la tabla de complejidad, que se quedan fuera del alcance de este libro.

Como ejemplo de puntos de función, imaginemos un sistema los elementos y complejidad indicados en la tabla 2.2, los puntos de función no ajustados son la suma de la multiplicación del número de elementos de cada tipo multiplicado por su peso, es decir:

$$PF_{NoAjustados} = \sum_{i=1}^{15} numeroElementos_i \cdot peso_i \quad (1)$$

Una vez calculados los puntos de función no ajustados, se les aplica un factor de corrección, llamado *Ajuste de Complejidad Técnica* (*PCA*, *Processing Complexity Adjustment* que depende de 14 atributos como muestra la tabla 3. Estos factores deben ser evaluados en una escala entre 0 y 5, donde 0 significa que el factor es irrelevante para la aplicación y un valor de 5 significa que es un factor esencial. También existen guías para asignar el valor

Cuadro 3: Factores de complejidad técnica

1	Comunicaciones de datos	8	Actualización *on-line*
2	Datos o procesamiento distribuido	9	Procesamiento complejo
3	Objetivos de rendimiento	10	Reutilización
4	Configuración utilizada masivamente	11	Facilidad de instalación y conversión
5	Tasa de transacción	12	Facilidad de operación
6	Entrada de datos *on-line*	13	Puestos múltiples
7	Eficiencia para el usuario	14	Facilidad de cambio

de cada factor. A continuación, el valor se ajuste se calcula con la ecuación:

$$PCA = 0,65 + (0,01 \cdot \sum_{i=1}^{14} F) \tag{2}$$

Nótese que la suma de todos los factores tiene un rango entre 0 y 70, por lo que PCA, está comprendido entre 0,65 y 1,35, es decir, los puntos de función no ajustados pueden variar un $\pm 35\,\%$ dependiendo de la complejidad y que cada factor influye un $5\,\%$.

Una vez calculados los puntos de función ajustados, se estiman las líneas de código mediante tablas que muestran el número medio de líneas de código por punto de función al estilo de la tabla 4. Finalmente, el esfuerzo se puede calcular con la productividad pasada de la organización.

Cuadro 4: Ejemplo de lenguajes y número medio de LoC por Punto de Función

Lenguaje de programación	*Media LoC/FP*
Lenguaje ensamblador	320
C	128
Cobol	105
Fortran	105
Pascal	90
Ada	70
Lenguajes orientados a objetos	30
Lenguajes de cuarta generación	20
Generadores de código	15
Hojas de cálculo	6
Lenguajes gráficos	4

Ejemplo

Continuando con el ejemplo anterior, para calcular los puntos de función ajustados y suponiendo que tenemos que la suma del factor de complejidad es 52, entonces calculamos el factor de ajuste (PCA):
$PCA = 0{,}65 + (0{,}01 \sum_{i=1}^{14} F_i) = 1{,}17$
aplicamos ese factor de ajuste obtenido a los puntos de función no ajustados:
$FP_{ajustados} = FPnoAjustados \cdot PCA = 372$
Después, basándonos en la tabla 4, sabemos que aproximadamente 128 líneas de código en `C` equivalen a un *punto de función*. Por tanto, tenemos:
$LoC = 128 \times FP = 47{,}616 LoC$ de código en `C`.
El esfuerzo se puede calcular como
$esfuerzo = FP/productividad$,
por lo que asumiendo que la productividad pasada de la organización fue de 12 $FP/personas - mes$
$Esfuerzo = 372/12 = 58 personas - mes$
y si la media es de 3.000 euros por $persona - mes$ (250 euros por FP) el coste será de 290.000 euros aproximadamente.

2.3. Modelos algorítmicos o paramétricos

Los modelos algorítmicos o paramétricos representan relaciones (ecuaciones) entre variables, generalmente esfuerzo y tamaño del software, pero pueden incluir otras características del proyecto como la complejidad o factores relacionados con la organización del proyecto.

Existen multitud de modelos paramétricos, que se pueden agrupar en (i) modelos abiertos, públicos o de caja blanca, si se conocen las ecuaciones que los componen o (ii) propietarios, cerrados o de caja negra, donde empresas comercializan las herramientas de estimación sin que se conozcan las ecuaciones.

Entre los primeros, tenemos las técnicas estadísticas de regresión, SLIM, COCOMO, Puntos de Función Albretch o sus variantes posteriores. Estos modelos se analizan más detalladamente a continuación. Entre los modelos propietarios podemos destacar PRICE-S, SEER-SEM o CHECKPOINT, aunque dada su naturaleza no son objeto de estudio en este libro.

2.3.1. Estimación mediante regresión estadística

Si se dispone de un histórico de proyectos, este método relaciona el coste o esfuerzo con otros parámetros del proyecto como tamaño, complejidad, etc utilizando algún tipo de curva de regresión. La figura 2 muestra dos ejemplo de curvas de regresión, donde el esfuerzo (e) se relaciona con el tamaño (t). Hay muchos tipos diferentes de regresiones matemáticas como por ejemplo regresión lineal simple, multivariable, exponencial, etc. que ajustan los datos disponibles a distintos tipos de curvas. Aunque es el modelo paramétrico más simple, hay estudios que muestran que pueden ofrecer buenas estimaciones.

2.3.2. COCOMO

El modelo paramétrico probablemente más conocido es el de COCOMO (*COnstructuctive COst Model*) al ser uno de los primeros modelos desar-

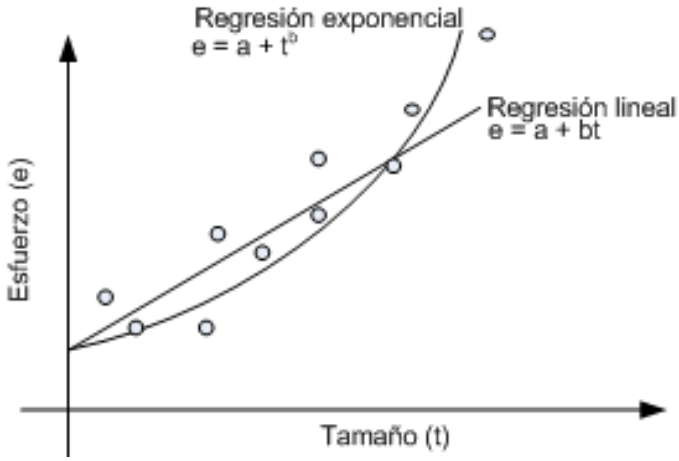

Figura 2: Ejemplos de curvas de regresión

rollados en los años 80 además de ser abierto, es decir, que se conocen las ecuaciones que lo componen. Fue propuesto por B. Boehm en 1981 empíricamente basándose en 63 proyectos principalmente de la NASA y actualizado en el año 1995 bajo la denominación de COCOMO II, que veremos más adelante.

En el modelo inicial COCOMO, también conocido como COCOMO 81, se distinguen tres tipos de proyecto según su dificultad y entorno de desarrollo:

- **Orgánico**. El proyecto se desarrolla en un entorno estable donde cambios en los requisitos son mínimos y el tamaño del proyecto es relativamente pequeño. Por ejemplo, los sistemas de gestión de la información.
- **Empotrado** (Embedded). El proyecto se desarrolla en un entorno con alta volatilidad de los requisitos, proyectos complejos o innovadores. Por ejemplo, proyectos en tiempo real en los que se desarrolla hardware y software (sistemas en tiempo real para aviones, telescopios, etc.).
- **Semi-libre** (Semidetached). Proyectos de desarrollo entre los modelo orgánico y empotrado. Ejemplos de proyecto de este tipo puede ser sistemas para automóviles como aparcamiento automático o *drivers* para hardware.

Cuadro 5: Parámetros de COCOMO básico e intermedio

	a Básico	a Intermedio	b	c	d
Orgánico	2,4	3,2	1,05	2,5	0,38
Semilibre	3,0	3,0	1,12	2,5	0,35
Empotrado	3,6	2,8	1,2	2,05	0,32

El modelo COCOMO se compone de 3 submodelos según la etapa de desarrollo en las que se encuentre el proyecto:

- **Básico**. Las ecuaciones de estimación de esfuerzo y tiempo de desarrollo sólo tienen en cuenta el tamaño del producto, para las estimaciones iniciales del proyecto. La medida del tamaño se basa en el número de líneas de código medidas en miles de instrucciones entregadas (KDSI):

  - $esfuerzo = a \cdot KDSI^b$
  - $tiempo = c \cdot esfuerzo^d$
  - $personal = esfuerzo/tiempo$

  donde los parámetros a, b y c dependen del tipo de proyecto (orgánico, semilibre o empotrado) como se muestra en la tabla 5. En COCOMO, por defecto, el *esfuerzo* se mide en horas trabajadas en un mes, considerando que un mes son 152 horas de trabajo. Por tanto, el *tiempo* se mide en meses. El *personal* sería el número de empleados a tiempo completo necesarios para llevar a cabo un proyecto.

Ejemplo COCOMO Básico
Asumamos que tenemos que desarrollar un sistema de facturación del cual tenemos experiencia (por tanto lo clasificamos como Orgánico). Además estimamos que el número de líneas de código es de 43,200. Aplicando las ecuaciones de COCOMO básico, obtenemos lo siguiente: • $esfuerzo = a \cdot size^b = 2,4 \cdot 43,2^{1,05} = 125,16$ persona/mes • $tiempo = c \cdot effort^d = 2,5 \cdot 94,93^{0,38} = 15,67$ meses • $personal = esfuerzo \div tiempo =\sim 8$ pers. (a tiempo completo)

- **Intermedio**. A la ecuación del modelo básico se incorpora un valor de ajuste compuesto por 15 atributos (llamados *cost drivers* en inglés) agrupados en 4 categorías: (i) características del producto, (ii) restricciones relacionadas con el hardware, (iii) niveles de experiencia del personal y (iv) características del proyecto en si mismo, es decir:

  - $esfuerzo_{intermedio} = esfuerzo_{nominal} \cdot EAF$

  donde EAF (*Effort Adjustment Factor*) es la multiplicación de los 15 factores con los parámetros de la tabla 6. La tabla indica porcentajes como por ejemplo, sí la capacidad de los programadores es muy baja (PCAP) añadirá un 42 % al esfuerzo de desarrollo, pero si por el contrario, la capacidad es muy alta, puede reducir el esfuerzo en un 30 % ($esfuerzo_{nominal} \cdot 0,70$).

  Continuando con el ejemplo anterior, y asumiendo que tenemos en cuenta el factor de ajuste con los valores de la tabla 2.3.2:

  - $esfuerzo_{intermedio} = esfuerzo_{nominal} \cdot EAF =$
    $= (3,2 \cdot 33,2^{1,05}) \cdot 0,608 = 76,96$ p/m
  - $tiempo = 2,5 \cdot 76,96^{0,38} = 13,02$ meses
  - $personal =\sim 6$ personas

Cuadro 6: Factores de COCOMO Intermedio

Factor	VL	L	N	H	VH	EH
Atributos del producto						
RELY Fiabilidad requerida del software	0,75	0,88	1	1,15	1,4	
DATA Tamaño de la base de datos		0,94	1	1,08	1,16	
CPLX Complejidad del producto	0,7	0,85	1	1,15	1,3	1,65
Atributos hardware						
TIME Restricciones en tiempo de ejecución			1	1,11	1,3	1,66
STOR Restricciones de memoria			1	1,06	1,21	1,56
VIRT Volatilidad entorno máquina virtual		0,87	1	1,15	1,3	
TURN Tiempo de ejecución completa		0,87	1	1,07	1,15	
Atributos del personal						
ACAP Capacidad de los analistas	1,46	1,19	1	0,86	0,71	
AEXP Experiencia en el área de la aplicación	1,29	1,13	1	0,91	0,82	
PCAP Capacidad de los programadores	1,42	1,17	1	0,86	0,7	
VEXP Experiencia en la máquina virtual	1,21	1,1	1	0,9		
LEXP Experiencia con el leng. de prog.	1,14	1,07	1	0,95		
Atributos del projecto attributes						
TOOL Utilización de herramientas software	1,24	1,1	1	0,91	0,82	
MODP Prácticas modernas de programación	1,24	1,1	1	0,91	0,83	
SCED Tiempo de desarrollo requerido	1,23	1,08	1	1,04	1,1	

Cuadro 7: Ej. asignación de valores a los factores en COCOMO

RELY Fiabilidad requerida del software	H	1,15
DATA Tamaño de la base de datos	VH	1,16
CPLX Complejidad del producto	H	1,15
TIME Restricciones en tiempo de ejecución	H	1,11
STOR Restricciones de memoria	N	1.00
VIRT Volatilidad entorno máquina virtual	L	0,87
TURN Tiempo de ejecución completa	L	0,87
ACAP Capacidad de los analistas	H	0,86
AEXP Experiencia en el área de la aplicación	H	0,91
PCAP Capacidad de los programadores	H	0,86
VEXP Experiencia en la máquina virtual	H	0,90
LEXP Experiencia con el lenguaje de programación	H	0,95
MODP Prácticas modernas de programación	VH	0,82
TOOL Utilización de herramientas software	N	1,00
SCED Tiempo de desarrollo requerido	N	1,00
EAF		**0,608**

- **Detallado**. Similar al intermedio, pero las estimaciones se realizan 4 fases del ciclo de vida: (i) *diseño de producto*, (ii) *diseño detallado*, (iii) *codificación y pruebas unitarias* e (iv) integración y *pruebas de integración*.

Finalmente, ya que el número de personas necesario en cada fase del ciclo de vida puede variar, se pueden utilizar perfiles predefinidos para la distribución de personal.

2.3.3. COCOMO II

En COCOMO II se reconocen varias cuestiones problemáticas que se estudiaron sobre el COCOMO original. Primero los procesos han evolucionado desde el modelo en cascada a otros modelos iterativos e inclementales, desarrollo basado en componentes, etc. Segundo, la cantidad de información disponible en las distintas etapas de desarrollo. Por ejemplo, el personal, características específicas del proyecto y que el problema de que la estimación de las líneas de código en las etapas tempranas del desarrollo es extremadamente difícil de calcular (tanto como el esfuerzo). Por tanto, según la información disponible dependiendo de las etapas de desarrollo, las estimaciones se hacen más acertadas (figura 3).

Por ello, sus autores lo actualizaron bajo la denominación COCOMO II. El nuevo modelo consta de de tres submodelos diferentes que tienen en cuenta en cuenta la información disponible en tres fases distintas del del desarrollo:

- **Fase 1. Composición de aplicaciones**. Este submodelo es usado para estimar el esfuerzo y el tiempo de desarrollo en proyectos en los que se utilizan prototipos para disminuir el riesgo de temas relacionados con la interfaz gráfica (GUI - *Graphical User Interfaces*) donde generalmente se utilizan herramientas CASE (Computer Aided Software Engineering) para desarrollo rápido de aplicaciones.

  Este modelo utiliza como variable de medida del tamaño del producto los Puntos Objeto que se basan en la el conteo artefactos como el

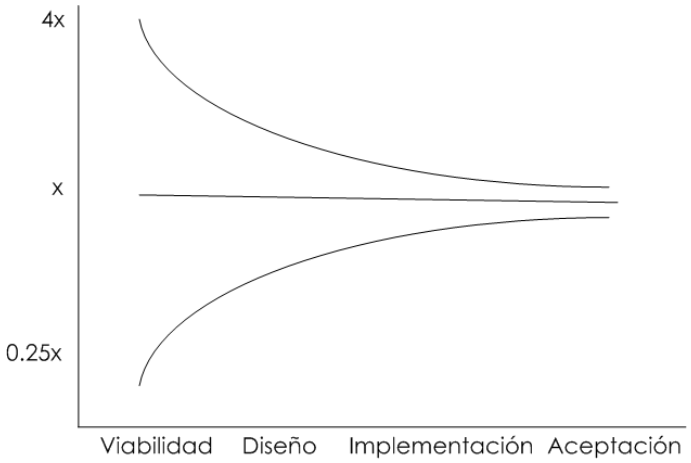

Figura 3: Variabilidad en las estimaciones según la fase de desarrollo (Boehm)

número de pantallas, informes y módulos. Este conteo es ponderado mediante un factor de complejidad compuesto de tres niveles, simple, medio y complejo.

- **Fase 2. Diseño preliminar**. En esta fase se tiene en cuenta la exploración de diferentes arquitecturas del sistema y conceptos de operación. La estimación de las líneas de código se realiza utilizando puntos de función (se ven a continuación). Utilizando una ecuación similar a COCOMO 81:

  $PM_{nominal} = A \cdot tamaño^B$

  pero en vez de considerar parámetros constantes para el exponente B como se hacía en COCOMO 81 con orgánico, semi-libre y empotrado, el exponente B se calcula mediante la ecuación (también utilizado en post-arquitectura):

  $B = 0,91 + 0,01 \sum W_i$

  donde cada W_i es la asignación de 0 a 5 (clasificados como {Muy Bajo, Bajo, Nominal, Alto, Muy Alto y Extra Alto}) a los siguientes factores de escala:

  - Precedentes (PREC). Considera la experiencia de la organización

en el desarrollo de aplicaciones similares.

- Flexibilidad (FLEX). Considera la rigidez de los requisitos y de las restricciones en el desarrollo.
- Arquitectura / Solución de riesgos (RESL). Tiene en cuenta las medidas tomadas para la eliminación de riesgos.
- Cohesión del equipo (TEAM). Considera las dificultades de cohesión y sincronización de todos las personas involucradas con el proyecto (usuarios, desarrolladores, equipo de pruebas, etc.
- Madurez de procesos (PMAT). Considera el nivel de madurez de la organización, basándose en el CMM (Capability Maturity Model).

Además, de manera similar al COCOMO Intermedio, se utilizan 7 factores de ajuste (EM):

$PM_{ajustados} = PMnominal(\prod_{i=1}^{7} EM_i)$

a los que les asigna un valor dependiendo de su clasificación en seis niveles {Muy Bajo, Bajo, Nominal, Alto, Muy Alto y Extra Alto}. Los factores de ajuste son:

- Capacidad del personal (PERS).
- Fiabilidad y complejidad del producto (RELY)
- Reutilización requerida (RUSE)
- Dificultad de la plataforma (PDIF)
- Experiencia del personal (PREX)
- Facilidades (FCIL)
- Calendario (SCED)

- **Fase 3. Post-arquitectura**. Una vez completado el diseño y definido la arquitectura, comienza el desarrollo. En este punto, la estimación puede hacerse bien mediante puntos de función o líneas de código, siendo similar al anterior, pero utilizando 17 factores agrupados cuatro categorías similares a COCOMO 81 y mostrados en la tabla 8.

Cuadro 8: Factores de COCOMO II

Producto
Fiabilidad requerida del software (RELY)
Tamaño de la base de datos (DATA)
Complejidad del producto (CPLX)
Reutilización requerida (RUSE)
Documentación desarrollada (DOCU)
Plataforma de desarrollo
Restricciones en el tiempo de ejecución (TIME)
Restricciones en el almacén principal (STOR)
Volatilidad de la plataforma (PVOL)
Personal
Capacidad de los analistas (ACAP)
Capacidad de los programadores (PCAP)
Experiencia en el desarrollo de aplicaciones similares (AEXP)
Experiencia con la plataforma de desarrollo (PEXP)
Experiencia con el lenguaje y herramientas (LEXP)
Estabilidad del personal (PCON)
Proyecto
Utilización de herramientas software (TOOL)
Desarrollo en múltiples localizaciones (SITE)
Tiempo necesario para el desarrollo (SCED)

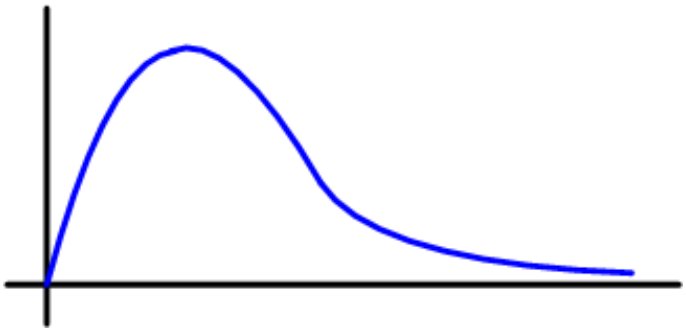

Figura 4: Distribución de Rayleigh

2.3.4. Putman – SLIM

Creado inicialmente por Putman sobre los años 1970 como modelo abierto, aunque posteriormente evolucionó en un producto comercial bajo el nombre de SLIM (*Software Life Cycle Model*) por la compañía SQM[1].

Este modelo se basa en que la distribución del personal en los proyectos se ajusta a la distribución de Norden-Rayleigh (figura 4), y a partir de esa distribución se hacen las estimaciones de esfuerzo y tiempo.

Putman definió una *ecuación del software*, basándose en que la cantidad de trabajo de un producto software es el producto de del esfuerzo realizado y el tiempo invertido. Matemáticamente se representa como:

$$S = C \cdot K \cdot t_d \tag{3}$$

donde S es el tamaño del producto medido en líneas de código, K el esfuerzo medido en personas-mes o personas-año, t_d el tiempo de desarrollo medido en meses o años, y C es una constante de productividad definida por un conjunto de prácticas de la organización tales como gestión del proyecto o del producto, tales como la complejidad. Utilizando su base de datos Putnam estableció la siguiente ecuación:

[1] `http://www.qsm.com/`

$$S = C \cdot K^{1/3} \cdot t_d^{4/3} \tag{4}$$

Por otra parte, para Putnam, la curva de utilización del personal en un proyecto, expresada mediante la curvas de Norden-Rayleigh permite definir una segunda ecuación para el esfuerzo y el tiempo. Basándose en proyectos de su organización, Putnam estableció el parámetro *acumulación de personal* (MBI – *Manpower Buildup Index*) definido como $MBI = K/t_d^3$, sugiriendo que la contratación de personal tiene implicaciones en el tiempo de desarrollo (t_d) y en el esfuerzo y reflejando que variaciones en los plazos afectan enormemente al esfuerzo. El MBI se puede definir y establecer previamente al desarrollo del proyecto, con los datos que la organización haya recogido. Utilizando las dos ecuaciones se pueden encontrar las diferentes combinaciones de tiempo y esfuerzo para definir la planificación.

Hay estudios que demuestran que SLIM no es método de estimación adecuado a pequeños proyectos, pero sí que podría serlo cuando el tamaño del proyecto es superior a 5,000 líneas de código, un esfuerzo superior 1,5 personas-año y tiempos de desarrollo superiores a 6 meses.

Hay estudios que demuestran que SLIM no es método de estimación adecuado a pequeños proyectos, pero sí que lo es cuando el tamaño del proyecto es superior a 5,000 líneas de código, un esfuerzo superior 1,5 personas-año y tiempos de desarrollo superiores a 6 meses.

2.4. Otros modelos

2.4.1. Modelos basados en la inteligencia artificial

Recientemente se han aplicado distintas técnicas de la minería de datos a distintos problemas de la ingeniería del software en general y a la estimación en particular. Entre este tipo de técnicas tenemos:

- **Sistemas basados en reglas**: Un sistema basado en reglas consiste en un conjunto de reglas con la siguiente forma:

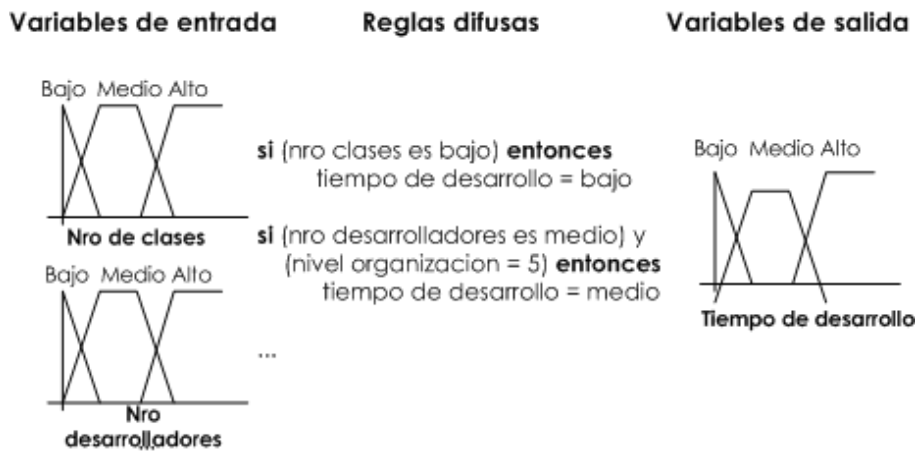

Figura 5: Ejemplo simplificado de utilización de reglas con lógica difusa

- **if** (30 $\leq$ CasosUso $\leq$ 35) **then** esfuerzo=50 personas/mes
- **if** (150 $\leq$ LoC_Clase) **then** esfuerzoPruebas=alto

Las reglas, generadas del histórico de proyectos mediante algoritmos de minería de datos, son una forma sencilla e intuitiva de obtener información y tomar acciones acordes a la información disponible en un momento concreto, pero también tienen el inconveniente de que al incrementar el número de reglas, su manejo y su significado de forma global puede ser difícil. Para entornos no determinísticos, es como el de la ingeniería del software donde existe incertidumbre, a las reglas se les ha incorporado lógica difusa. La figura 6 muestra un ejemplo muy simplificado de uso de reglas con lógica difusa, donde se les asignan valores lingüísticos del estilo, pequeño, mediano y grande a variables del proyecto; una vez obtenidas las reglas mediante algoritmos de inteligencia artificial o minería de datos, estas se pueden utilizar para la realización de estimaciones o tomar acciones correctivas cuando el proyecto ya esta en marcha.

- **Razonamiento basado en casos** Este método consiste en replicar el proceso que un experto llevaría a cabo para tomar una decisión basándose en su experiencia (ver método Delphi en la sección 2.1).

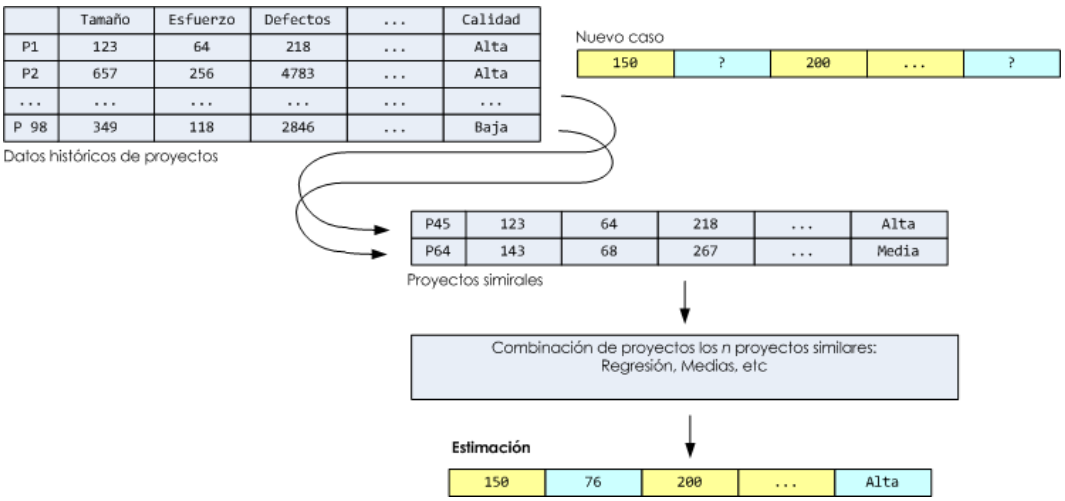

Figura 6: Proceso de estimación por analogía

Teniendo datos históricos de proyectos, una nueva estimación se obtiene encontrando un número proyectos similares conforme a los datos conocidos del proyecto a estimar (basándose en alguna medida de distancia como la distancia euclídea). Después, con ese reducido número de proyectos, se estiman los parámetros desconocidos haciendo por ejemplo la media, regresiones o similar. La figura 6 muestra el proceso de estimación basado en casos.

- **Redes neuronales** Generalmente usados para reconocimiento de patrones y clasificación, las redes neuronales también han sido utilizadas en la ingeniería del software. Como el ejemplo mostrado en la figura 7, una red neuronal consiste en un conjunto de nodos, generalmente organizados en capas (una capa de entrada, una o dos capas intermedias y una capa de salida) y unas conexiones con sus pesos de las que depende la ïnformación"que se propaga. Existe multitud de tipos de redes neuronales, siendo el más conocido el *perceptron*, también mostrado en la figura 7, el que la información se va propagando hacia adelante, de la capa de entrada compuesta por atributos tales como el número de casos de uso, experiencia del personal, etc. hacia la capa de salida con los parámetros que se quieren estimar tales como esfuerzo, tiempo de

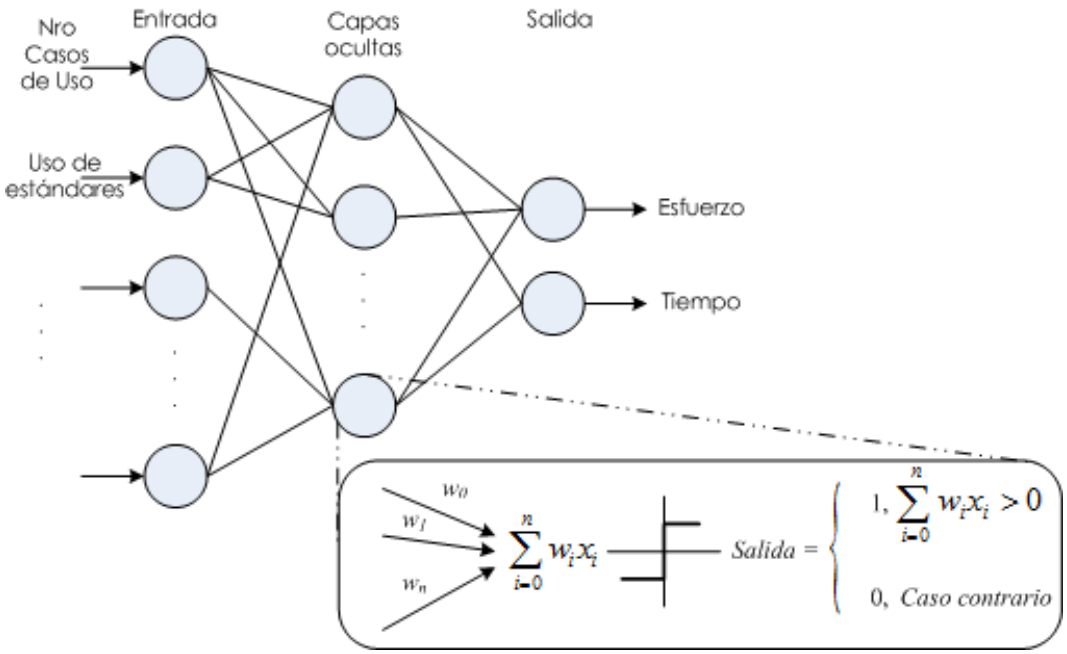

Figura 7: Proceso de estimación por analogía

desarrollo, etc. El mayor inconveniente para la ingeniería del software de las redes neuronales, es que una vez entrenadas, i.e, los pesos se han ajustado con los datos de entrenamiento proporcionados, esas actúan como cajas negras en las que no se sabe el "proceso"por el que se ha llegado a dicha estimación. Otro posible inconveniente, es que si no se tiene cuidado en la forma de realizar el aprendizaje, las redes pueden sobre-aprender (conocido con el termino inglés de *overfitting*) los datos de entrenamienten y no generalizar correctamente.

- **Árboles de decisión** En general, los árboles de decisión son utilizados para predecir o explicar observaciones. Las hojas de los árboles de decisión pueden ser valores discretos (o etiquetas), entonces llamados árboles de clasificación o ecuaciones de regresión, a los cuales también se les llama árboles de regresión. Los nodos intermedios representan decisiones, siendo los nodos más altos en la jerarquía, los más importantes. Existen diferentes algoritmos para la creación de árboles de decisión

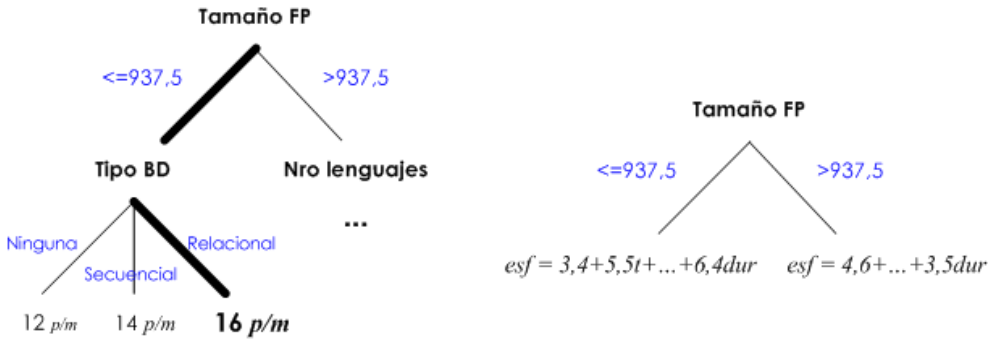

Figura 8: Ejemplo de árbol de decisión

y una de sus ventajas es su directa conversión a reglas que pueden ser fácilmente interpretadas por los gestores de proyectos. La figura 9 muestra dos árboles uno donde las hojas son valores concretos (árbol de clasificación) y otro donde las hojas son ecuaciones de regresión (árbol de regresión) para la estimación del esfuerzo. Por ejemplo, en el árbol de la izquierda el principal atributo es el tamaño, y dependiendo si es menor que esa cantidad, se comprueba el segundo atributo (tipo de base de datos), y si este es relacional la estimacion del esfuerzo sería 16 personas mes. En el árbol de la derecha, se aplicaría una ecuación u otra dependiendo también del tamaño.

- **Otras técnicas de inteligencia artificial** Además de las técnicas mencionadas, otras técnica basadas en la inteligencia artificial y minería de datos que han sido aplicadas a la ingeniería del software incluyen los algoritmos genéticos para encontrar ecuaciones de por ejemplo predicción de esfuerzo y las redes bayesianas como técnica probabilística. Además algunas técnicas pueden combinarse como forma de mejorar las estimaciones.

De todas formas, cabe destacar la estimación es un problema abierto. Hasta la fecha, parece ser que la utilización de técnicas complejas, no mejoran sustancialmente las estimaciones cuando se comparan con otras más simples. Si parece que el uso de expertos humanos en conjunción con alguna otra

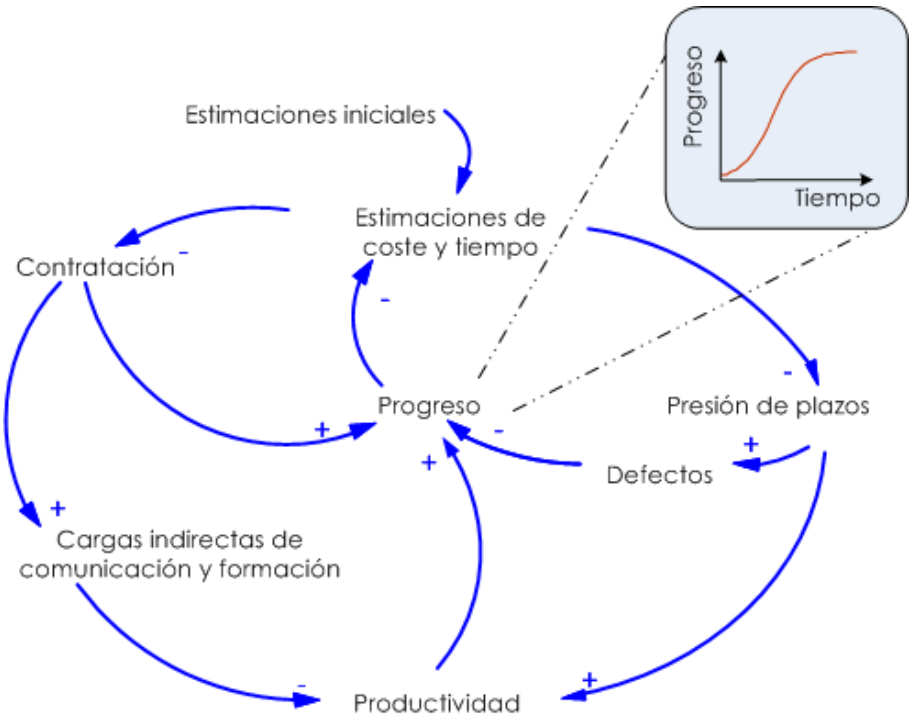

Figura 9: Ejemplo de árbol de decisión

técnica es superior a su uso individual.

2.4.2. Sistemas dinámicos

Los modelos dinámicos iniciados por J. Forrester en los años 1960, permiten estudiar cómo los sistemas complejos evolucionan en el tiempo. Mediante bucles causales donde ciertas variables influyen positivamente o negativamente en otras, permiten simular comportamientos en el tiempo de sistemas complejos.

En relación a la gestión en la ingeniería del software, los sistemas dinámicos permiten la simulación del comportamiento proyectos software, de tal manera que los gestores de proyectos pueden simular la ejecución de un proyecto con diferentes parámetros, analizando así las políticas de gestión que mejor se adaptan a los objetivos del proyecto. Además las simulaciones

pueden realizarse en distintas etapas en el desarrollo de un proyecto:

- Análisis a priori del proyecto para la estimación y prever comportamientos. Por ejemplo, preguntas del estilo: ¿qué ocurrirá si se añaden 3 personas a mitad de proyecto?
- Control del proyecto durante su ejecución para adaptar las estimaciones a la evolución del proyecto. Por ejemplo, preguntas del estilo ¿qué está ocurriendo con la calidad con los actuales parámetros?
- Análisis *post-mortem* del proyecto finalizado con la idea de mejorar el desarrollo de futuros proyectos para conocer cómo podrían haberse mejorado los resultados si se hubieran aplicado otros parámetros, por ejemplo con preguntas tales como ¿qué habría ocurrido si se hubiera incorporado 2 personas más al equipo de pruebas?

Al igual que otras técnicas de estimación, los modelos de sistemas dinámicos necesitan ser validados y adaptados a las organizaciones donde se utilizan, ya que la evolución de los mismo depende de las estimaciones iniciales, políticas de gestión, características del proyecto y características de la organización.

Entre los modelos de sistemas dinámicos más conocidos y uno de los primeros, se encuentra el modelo de Abdel-Hamid. Este se compone de cuatro diferentes submodelos con las actividades relacionadas con la gestión de recursos humanos, la producción, la planificación y el control de proyectos software, cada una de ellas con un gran número de parametros. Este modelo ha sido ampliamente validado, utilizado y extendido posteriormente.

2.5. Evaluación de modelos

Aunque no se puede decir que existan métodos de estimación superiores a otros, a una misma entrada, los distintos métodos de estimación tales como la regresión, COCOMO, puntos de función, redes neuronales, programación

genética, etc. generarán diferentes resultados. Además puede variar la información disponible a la hora de utilizarlos tales como la base de datos de históricos de proyectos, el calibrado realizado o la experiencia del personal con los métodos, etc.

Las técnicas más usadas en la ingeniería del software para determinar bondad de la estimación y de la capacidad de predicción de una técnica son el *Nivel de Predicción* y la *Magnitud Media del Error Relativo*. Además con las curvas de regresión, es típico utilizar las técnicas estadísticas del coeficiente de correlación (R) y determinación (R^2).

- *Magnitud Media del Error Relativo*, $MMRE$, se define como:

$$MMRE = \frac{1}{n}\sum_{i=1}^{n}|\frac{e_i - \hat{e}_i}{e}| \tag{5}$$

  donde e es el valor real de la estimación, $\hat{e}$ es el valor estimado y n es el número de proyectos. Cuanto menor es $MMRE$, mejores son predicciones del modelo evaluado. Un criterio habitual en la ingeniería del software, es que $MMRE$ sea menor que 0,25 para considerar el modelo como bueno. Como ejemplo, la figura 10 muestra gráficamente las distancias entre los valores reales y los estimados (la curva regresión) cuyas distancias son las utilizadas para el calculo de $MMRE$. Supongamos un transcurridos 9 proyectos en una organización obtenemos los resultados de la tabla 10 cuyo valor de $MMRE$ tal y como muestra la ecuación 5 sería:

  $MMRE = \frac{1}{n}\sum_{i=1}^{n}|\frac{e_i-\hat{e_i}}{e}| = \frac{1}{9}(\frac{|145-120|}{120} + \frac{|250-320|}{320} + ... + \frac{|320-260|}{320}) = 0,20$

  por tanto, podemos decir que el método de estimación satisface el criterio de $MMRE \leq 0,25$.

- *Nivel de predicción* l, $Pred(l)$, donde l es un porcentaje, se define como el cociente entre el número de estimaciones que están dentro del porcentaje l en valor absoluto entre el número total de estimaciones. Por ejemplo, $PRED(0,2) = 0,8$ quiere decir que el 80 % de las esti-

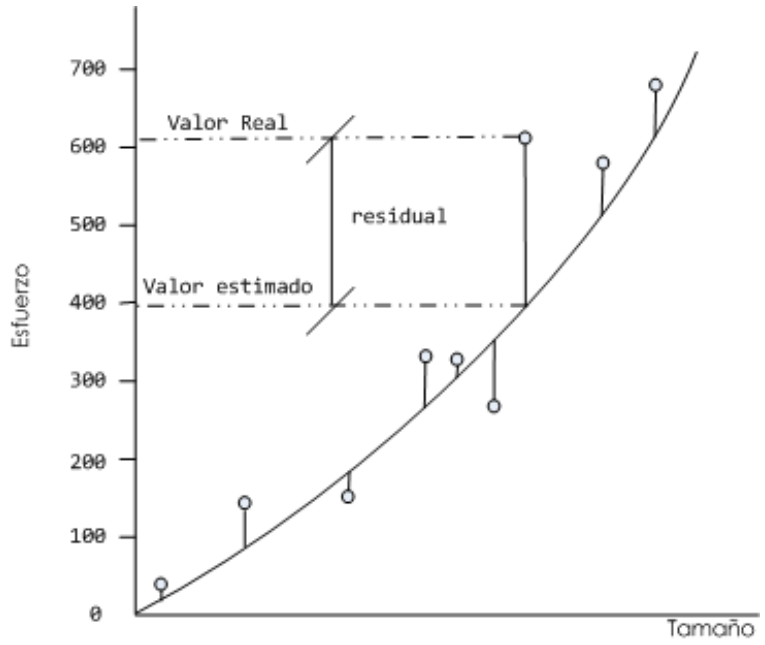

Figura 10: MMRE como indicador de la distancia entre los valores reales y estimados

maciones están dentro del 20 % de las estimaciones reales. En la ingeniería del software, el criterio habitual para aceptar un modelo como bueno suele ser que $Pred(0,25) \geq 0,75$, es decir, el 75 % de las estimaciones se desvían menos del 25 % del valor real. La figura 11 representa gráficamente la *Nivel de predicción*, donde los trazos verticales junto a los puntos de estimación muestran la variaciones de la estimación al $\pm 25\,\%$; el valor $Pred(25)$ se calcula sumando el número de veces que la curva corta con los trazos verticales, y después dividiendo esa suma entre el número total de estimaciones. Si continuamos con el ejemplo de la tabla 10, tenemos que 7 de los 9 proyectos están dentro del 25 %, entonces:

$Pred(25) = \frac{7}{9} = 0,77$

Por tanto, como $MMRE < 0,25$ y $Pred(25) > 0,75$, se podría considerar que el método de estimación es aceptable.

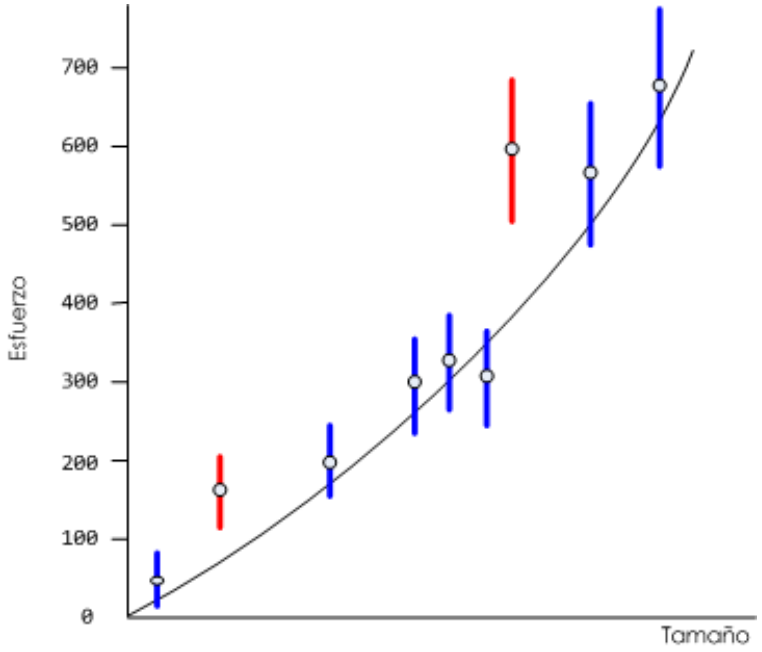

Figura 11: Ejemplo de Predicción de Nivel, l, $Pred(l)$

- *Coeficiente de determinación múltiple*, R^2, es una medida común en el análisis de regresión denotando como ajustan los datos a la recta de regresión. Si todas los puntos coinciden con la recta de regresión, es decir si hay una relación entre las variables dependiente e independiente perfecta, el valor de R^2 es 1, y según los puntos se alejan el valor disminuye hasta que si no hay relación alguna entre las variables, el valor de R^2 es 0. Matemáticamente se define como:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(e_i - \hat{e}_i)^2}{\sum_{i=1}^{n}(e_i - \bar{e})^2} \tag{6}$$

  donde dados n estimaciones de proyectos, e_i es el valor real, $\hat{e}_i$ es el valor estimado y $\bar{e}$ es la media.

Cuadro 9: Ejemplo de valoración de proyectos

	Estimado	Real	|Diferencia|	±25 % Estimado	% Error	Error < 25 %
A	120	145	25	90-150	0,20	Cierto
B	235	320	85	187,5-312,5	0,26	Falso
C	60	50	10	45-75	0,16	Cierto
D	100	120	20	75-125	0,20	Cierto
E	400	610	210	300-500	0,52	Falso
F	500	580	80	375-625	0,16	Cierto
G	280	310	30	210-350	0,10	Cierto
H	670	690	20	502,5-837,5	0,02	Cierto
I	320	260	60	240-400	0,18	Cierto

Calibración de modelos

Una característica tanto de los modelos de estimación paramétricos o los basados en la minería de datos es que necesitan ser calibrados y ajustados a los entornos en los que se utilizan. Por ejemplo, COCOMO 81 se definió utilizando 63 proyectos, pero es sabido que utilizando los parámetros por defecto las estimaciones son bastante pobres debido a las diferencias en los entornos, tecnologías, etc. Por lo tanto, según una organización va recabando datos de los proyectos que realiza, los modelos utilizados en la estimación deben calibrarse con la nueva información, siguiendo los siguiente pasos:

- Lo primero es verificar que los nuevos datos, las métricas, son consistentes con la definición utilizada en el modelo a calibrar. También puede ser necesario quitar datos que han dejado de ser útiles para la organización, por ejemplo, datos de proyectos de tecnologías obsoletas que la organización no volverá a realizar.
- Después se calibra el modelo ajustándolo a los datos, por ejemplo, los parámetros en el caso de las curvas de regresión o ejecutando los algoritmos de aprendizaje con los datos nuevos. Los nuevos modelos se pueden validar con las medidas mencionadas en la sección anterior.
- Finalmente se reemplaza en viejo modelo con el nuevo.

3. Planificación y seguimiento del proyectos

La planificación de un proyecto software, al igual que cualquier otro tipo de proyecto, consiste en identificar las tareas del proyecto, las relaciones entre las tareas (en qué orden deben de ser ejecutadas), asignar los recursos a las tareas y estimar los plazos de ejecución de las tareas.

En la planificación de tareas, el proyecto se divide generalmente en subtareas de forma jerárquica, lo que se conoce como la estructura de descomposición del trabajo, o en sus siglas en inglés, WBS (*Work Breakdown Structure*). Una vez definidas las tareas, debe determinarse cuales son las relaciones entre ellas y el orden en que se deben llevar a cabo. Para ello frecuentemente se hace uso de técnicas tales como CPM (*Critical Path Method*), PERT (*Program Evaluation and Review Technique*) y Gantt. A las actividades se le debe de asignar los recursos necesarios de personal, espacio, etc. de la manera más eficiente posible procurando que las tareas no necesiten los mismos recursos al mismo tiempo para que estos estén disponibles. Una vez planificado el proyecto y según se vaya ejecutando el mismo, es necesario hacer un seguimiento para ir detectando las divergencias, actualizando la planificación añadiendo tareas olvidas, actualizando costes y realizando nuevas estimaciones, reasignando recursos, motivando al personal y todo tipo de inconvenientes que pueden ocurrir durante la realización del proyecto. En las siguientes secciones estudiaremos en más detalle estas técnicas.

3.1. Estructura de Descomposición del Trabajo

La Estructura de Descomposición del Trabajo (EDT), aunque más conocido por sus siglas en inglés WBS (*Work Breakdown Structure*), es una técnica para descomponer el proyecto en sus distintas tareas o paquetes de trabajo (*work packages*) de forma jerárquica. La organización de las tareas puede

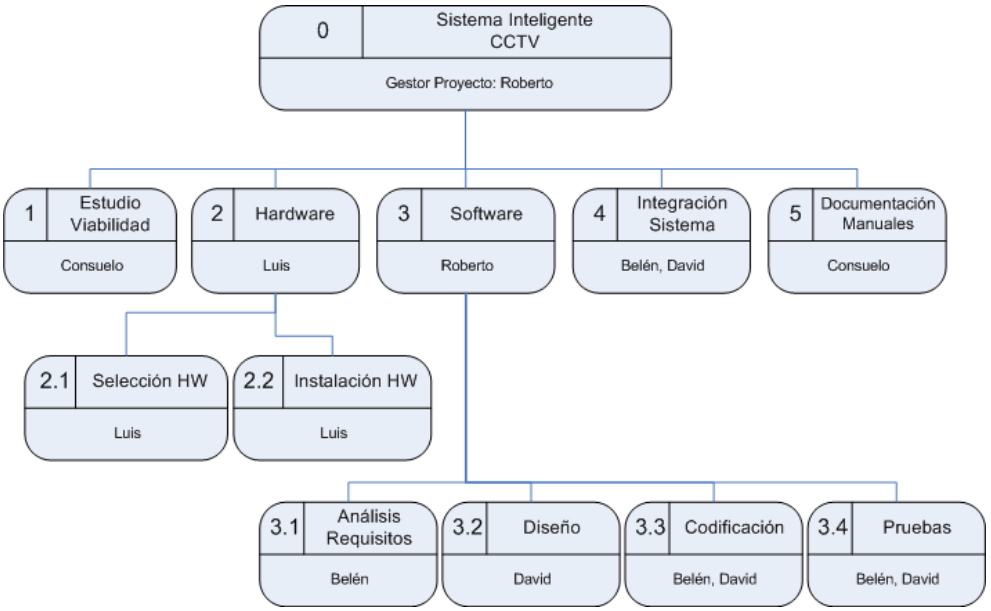

Figura 12: Ejemplo WBS

hacerse bien mediante la descomposición de los componentes o bien mediante las fases de trabajo. La figura 12 y la tabla 10 muestran un ejemplo de WBS, donde se puede apreciar que la jerarquía permite agrupar las tareas relacionadas.

Aunque en proyecto de cierta enjundia puede ser difícil, es importante asegurarse de que no se olvida ninguna tarea, ya que las tareas aquí definidas forman parte de la planificación posterior que se representarán las tareas gráficamente considerando el orden entre ellas (relaciones entre las tareas predecesoras y sucesoras).

3.2. El método del camino crítico y PERT

El método del camino crítico o análisis del camino crítico, más conocido por sus siglas en inglés CPM (*Critical Path Method*) y la técnica de evaluación y revisión de Proyectos, también más conocida por PERT (*Program Evaluation and Review Technique*) son técnicas de redes utilizadas para la

Cuadro 10: Actividades en la estructura WBS

Id	*Actividad*	*Responsable*
0	Sistema Inteligente CCTV	Roberto
1	Estudio viabilidad	Consuelo
2	Hardware	Luis
2.1	Selección HW	Luis
2.2	Instalación HW	Luis
3	Software	Roberto
3.1	Análisis requisitos	Belén
3.2	Diseño	David
3.3	Codificación	Belén, David
3.4	Pruebas	Belén, David
4	Integración	Belén, David
5	Documentación y manuales	Consuelo

gestión de proyectos. Como ya se ha mencionado anteriormente, todo proyecto se descompone en un número de actividades, generalmente representadas utilizando WBS, cada una de ellas tendrá una duración, y tendrán que ejecutarse en cierto orden, es decir, hay tareas predecesoras y sucesoras.

CPM y PERT son técnicas prácticamente equivales, desarrolladas por grupos diferentes y casi simultáneamente. PERT fue desarrollado en 1957 por la empresa Booz Allen Hamilton para el Departamento de Defensa de los de EE.UU. mientras que CPM fue desarrollado por la empresa química *du Pont* en 1958. La diferencia entre los dos es que CPM considera un valor único de estimación de la duración de una tarea (determinista) y PERT utiliza la estimación en 3-puntos (estadística) donde para cada estimación se pondera entre su valor pesimista, realista y optimista.

Ambos, CPM y PERT, se basan en representar el proyecto en forma de grafos para ordenar las actividades estableciendo sus dependencias y encontrar así el *camino crítico*, que denota el camino del grafo cuyas actividades no se puede retrasar si el proyecto ha de finalizar en el plazo establecido. O dicho al revés, si alguna de las actividades en este camino se retrasa, también se retrasará la finalización del proyecto. Existen dos representaciones equivalentes, que se denominan *diagrama de dependencias* y *diagrama de flechas*

dependiendo si son los arcos o los nodos los que representan a las actividades. En ambos se utiliza el siguiente proceso:

- Identificar las actividades y dependencias bien en un diagrama de flechas o diagrama de precedencia.
- Realizar el recorrido hacia adelante"para calcular las fechas más tempranas en las que una actividad puede ser iniciada y completada.
- Realizar el recorrido hacia atrás"para calcular las fechas más tardías en las que una actividad puede ser iniciada y completada.
- Calcular las holguras disponibles (a veces se utiliza el término inglés *slack*) de las actividades. Estas representan el tiempo que una actividad puede retrasarse, sin que se retrase la fecha de finalización del proyecto.
- Identificar el camino crítico, es decir, las actividades sin holgura y que por tanto, si alguna de estas actividades en el camino crítico se retrasase, también se retrasaría la fecha de finalización del proyecto.

A continuación se muestra el proceso mediante un ejemplo y utilizando la notación de diagrama de flechas. Imaginemos la estructura WBS de la tabla 11 en la que ya se ha definido que tareas son predecesoras de otras. A partir de dicha estructura de paquetes, el primer paso sería construir el grafo con las tareas como arcos, y los hitos (comienzo y fin de tareas) son los nodos. Es su construcción debemos tener en cuenta que sólo puede haber un nodo inicial y un nodo final (no hay nodos colgados) y además, no puede haber bucles. La figura 13 representa el diagrama de flechas ordenando las tareas donde cada nodo simplemente está numerado secuencialmente y en los arcos cada actividad tiene asociada su duración.

Después calculamos las siguientes fechas en dos pasadas (hacia adelante y hacia atrás):

- Inicio más temprano (ES – *Early Start Time*), es la fecha más próxima en la que una actividad puede comenzar.

Cuadro 11: Ejemplo CPM. Actividades con su duración y precedencias

Id		*Actividad*	*Duración*	*Precedencias*
0		Sistema Inteligente CCTV		
1	(A)	Estudio viabilidad	5	—
2		Hardware		
2.1	(B)	Selección HW	9	A
2.2	(C)	Instalación HW	11	B
3		Software		
3.1	(D)	Análisis de requisitos	8	A
3.2	(E)	Diseño	10	D
3.3	(F)	Codificación	18	E
3.4	(G)	Pruebas unitarias	7	F
4	(H)	Integración y pruebas del sistema	15	C, G
5	(I)	Documentación y manuales	25	—

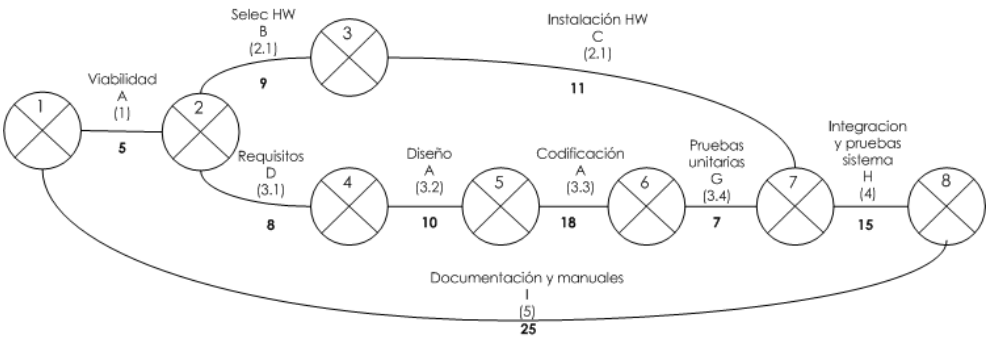

Figura 13: Ejemplo CPM. Diagrama de flechas inicial

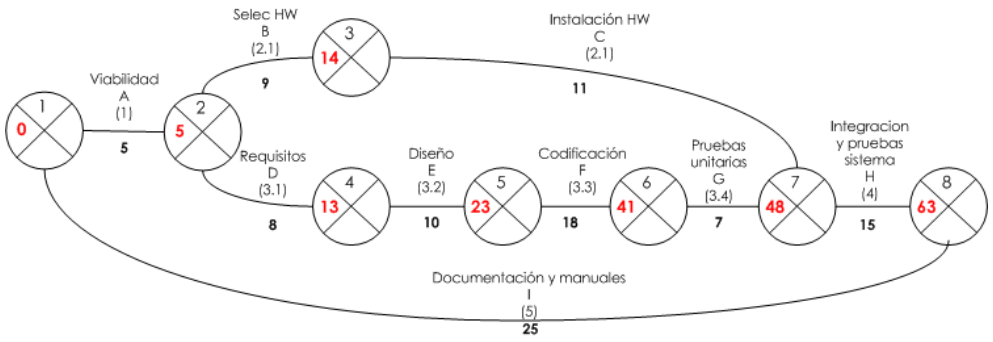

Figura 14: Ejemplo CPM. Recorrido hacia adelante

- Terminación más temprana (LS – *Late Start Time*), es la fecha más próxima en la una actividad puede terminar.
- Inicio más tardío (EF – *Early Finish Time*), es la fecha más lejana en el que una actividad puede comenzar sin que retrase el proyecto.
- Terminación más tardía (LF – *Late Finish Time*), es la fecha más lejana en que una actividad puede terminar sin retrasar la fecha de conclusión del proyecto.

En el **recorrido hacia adelante**, donde se va calculando de izquierda la derecha fecha de *inicio más temprano* que las actividades puede comenzar. Para ello, se inserta 0 como fecha inicial y se van sumando los tiempos de las actividades, cuando hay varias actividades precedentes que convergen en un nodo se escoge el máximo de los tiempos. La figura 14 muestra las fechas de inicio temprano del ejemplo anterior.

En el **recorrido hacia atrás**, donde se va calculando de derecha a izquierda la fecha de *terminación más tardía* que las actividades puede finalizar. Para ello, primero se copia la última fecha de *inicio más temprano* como *fin más tardío* en el último nodo (derecha) y se se va restando la duración de los tiempos de las actividades. Cuando hay varias actividades sucesoras en una nodo se escoge la mínima. La figura 15 muestra los resultados de la pasada hacia atrás.

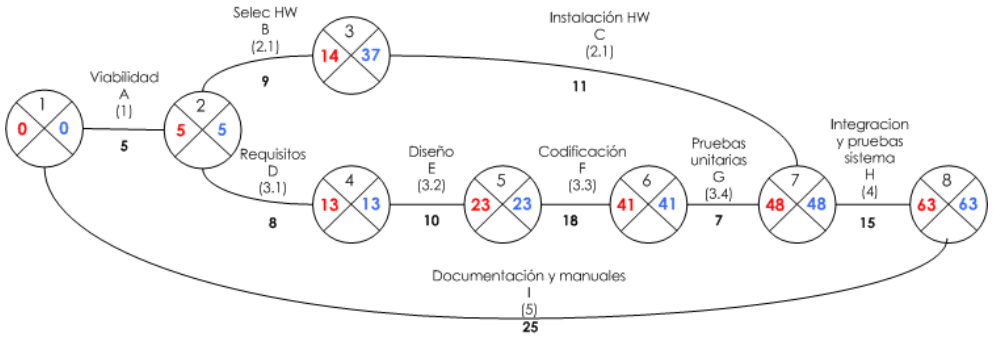

Figura 15: Ejemplo CPM. Recorrido hacia atrás

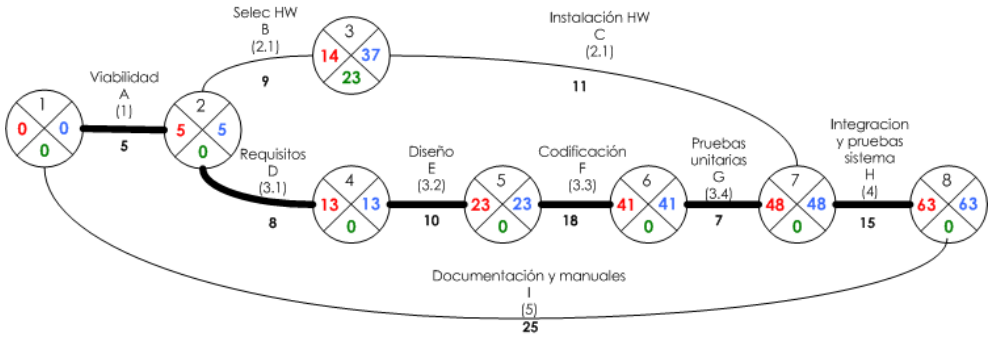

Figura 16: Ejemplo CPM. Holgura y camino crítico

La **holgura** disponible de las actividades se calcula restando el *inicio más temprano* (ES) y el *inicio más tardío* (LF), o la *terminación más temprana* (EF) y la *terminación más tardía* (LF). Aquellas actividades que no tengan holgura, es decir que las 2 fechas sean iguales, forman el *camino crítico*, es decir, aquellas actividades que no pueden retrasarse para que el proyecto no se retrase y por tanto, son las actividades que tenemos que vigilar con más detalle. La figura 16 muestra el cálculo de la holgura para el ejemplo, marcando el camino crítico con el trazo grueso. Podremos *jugar* con las las actividades que no estén en este camino, comenzándolas más tarde, rompiéndolas en dos partes, asignando menos recursos, etc. siempre y cuando no excedan el límite de la holgura. La tabla 12 muestra el resumen de todos los variables.

Finalmente, puede haber tareas ficticias (*dummy*), de duración cero, que

Cuadro 12: Valores de CPM

Act	Dur	ES	LS	EF	LF	Hol	Camino crítico?
A	5	0	0	5	5	0	Si
B	9	5	28	14	37	23	No
C	11	14	37	25	48	34	No
D	8	5	5	13	13	0	Si
E	10	13	13	23	23	0	Si
F	18	23	23	41	41	0	Si
G	7	41	41	48	48	0	Si
H	15	48	48	63	63	0	Si
I	25	0	38	25	63	38	No

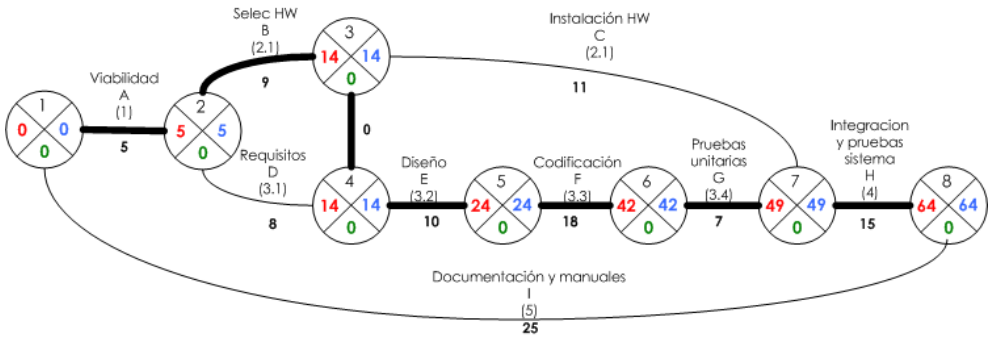

Figura 17: Holgura y camino crítico con una tarea ficticia

se utilizan para asignar precedencias en el orden de las tareas. Siguiendo el ejemplo anterior, si fuese necesario saber el tipo de hardware, es decir haber terminado la tarea "Selección HW", para realizar la tarea de "Diseño", tendríamos una precedencia más tal y como se indica en la tabla 13. El resultado final sería el mostrado la figura 17, donde se puede ver que entre los nodos 3 y 4 tenemos una tarea ficticia. Además podemos ver que cambia el camino crítico al igual que los cálculos de las fechas.

Cuadro 13: Actividades incluyendo una nueva precedencia

Id		*Actividad*	*Duración*	*Precedencias*
0		Sistema Inteligente CCTV		
1	(A)	Estudio viabilidad	5	—
2		Hardware		
2.1	(B)	Selección HW	9	A
2.2	(C)	Instalación HW	11	B
3		Software		
3.1	(D)	Análisis de requisitos	8	A
3.2	**(E)**	**Diseño**	**10**	**B, D**
3.3	(F)	Codificación	18	E
3.4	(G)	Pruebas unitarias	7	F
4	(H)	Integración y pruebas del sistema	15	C, G
5	(I)	Documentación y manuales	25	—

3.3. Diagramas de Gantt

Los diagramas de redes CPM y PERT definen las restricciones entre tareas pero no su calendario. Los diagramas de Gantt, inventados por Karol Adamiecki en 1896 y reinventados por Henry Gantt en 1910, muestran la evolución de las actividades con sus fechas e hitos. Una de las ventajas del los diagramas de Gantt es que facilitan la visión global del proyecto, mostrando el comienzo y fin de las actividades, la asignación de recursos a las tareas y durante el desarrollo del mismo, las desviaciones entre los plazos estimados y reales. Como muestra el ejemplo de la figura 18 en el eje vertical se muestran las actividades y en horizontal el tiempo. Otra ventaja de los diagramas de Gantt es que es una técnica fácil de comprender por todos los miembros del proyecto y además permite hacer un seguimiento del mismo.

Los diagramas de Gantt se suelen complementarse asociando recursos a las actividades, para monitorizar la carga de trabajo del personal y el progreso de las tareas (ver figura 19). Entre las desventajas se encuentran que si tenemos un gran número de tareas es más complicado de usar y ver la relación entre las tareas.

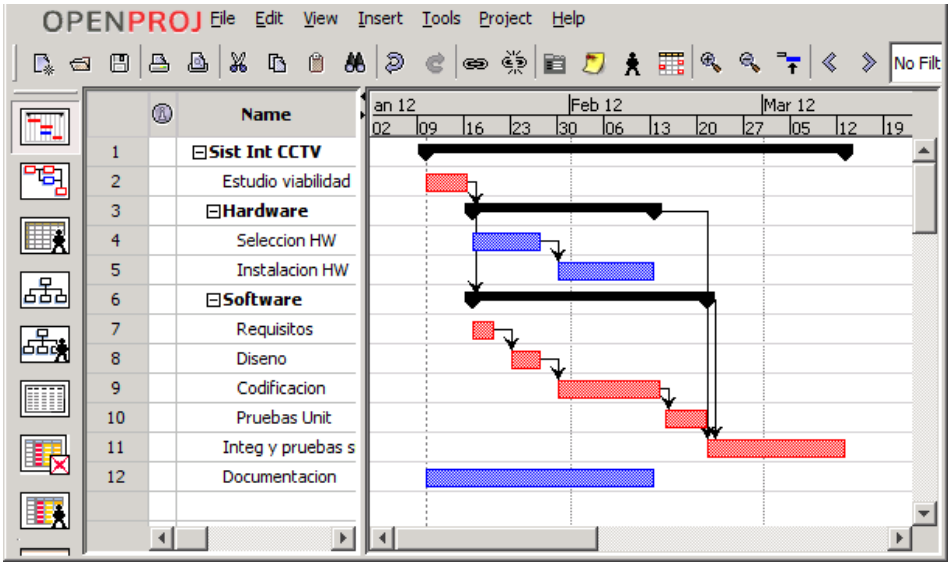

Figura 18: Ejemplo de diagrama de Gantt

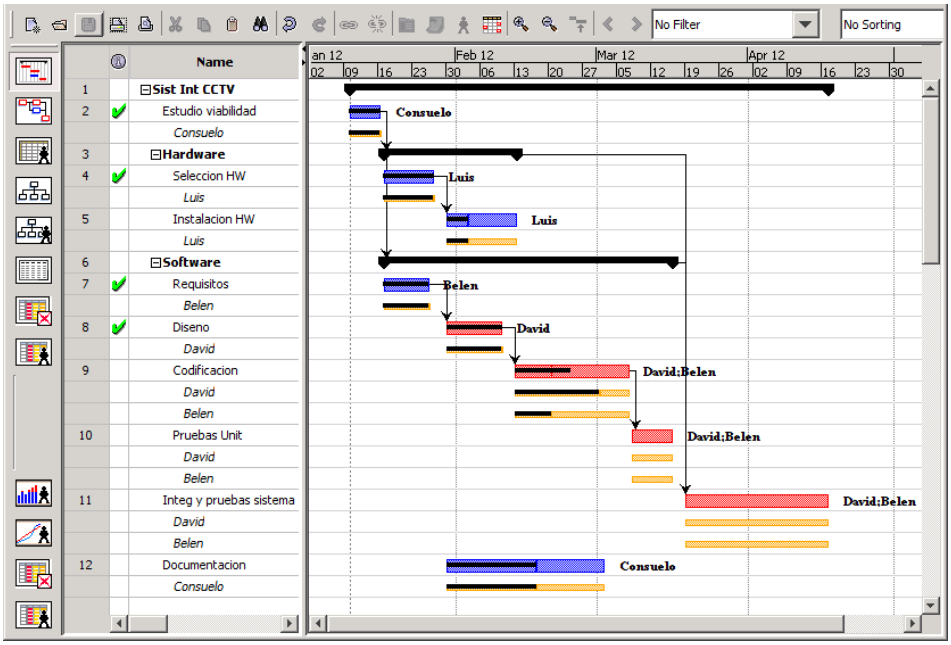

Figura 19: Ejemplo de diagrama de Gantt mostrando recursos y estado de las tareas

3.4. Método del valor conseguido

El *Sistema del Valor Conseguido*, más conocido por sus siglas en inglés, EVS (*Earned Value System*), se desarrollo en el Departamento de Defensa de los EE.UU. en los años 1960 para el control de proyectos. EVS ha sido incluido en el cuerpo del conocimiento de la gestión de proyectos (PMBOK) que a su vez ha sido adoptado como estándar IEEE 1490.

EVS permite hacer un seguimiento del proyecto integrando información de plazos y costes. Además permite visualizar esta información en una única gráfica mediante tres variables que muestran la evolución de un proyecto a lo largo del tiempo:

- *Coste Presupuestado* o Coste Presupuestado del Trabajo Planeado (BCWS – *Budgeted Cost of Work Scheduled*) se calcula sumando los costes planificados del proyecto.
- *Valor Actual* o Coste Real del Trabajo Realizado (ACWP – *Actual Cost of Work Performed*) representa el esfuerzo invertido hasta la fecha, es decir, los costes realmente gastados en las tareas llevadas a cabo en el proyecto hasta la fecha.
- Valor Conseguido (*Earned Value*) o Coste Presupuestado del Trabajo Realizado (BCWP – *Budgeted Cost of Work Performed*) representa el grado de finalización de las tareas. Se calcula sumando los costes planificados de las tareas que realmente se han llevado a cabo hasta la fecha.

A partir de estas variables, se puede calcular el estado actual del proyecto y predecir cómo irá su ejecución en el lo que quede de proyecto en una serie de indicadores. Estos indicadores tienen la finalidad de orientar a los gestores de proyectos a realizar acciones correctivas cuando los proyectos se desvían de la planificación. Además es posible representarlas gráficamente , y lo que es más importante. La figura 6 muestra cómo dichos indicadores se pueden combinar en una única gráfica, con lo que los gestores de proyectos pueden

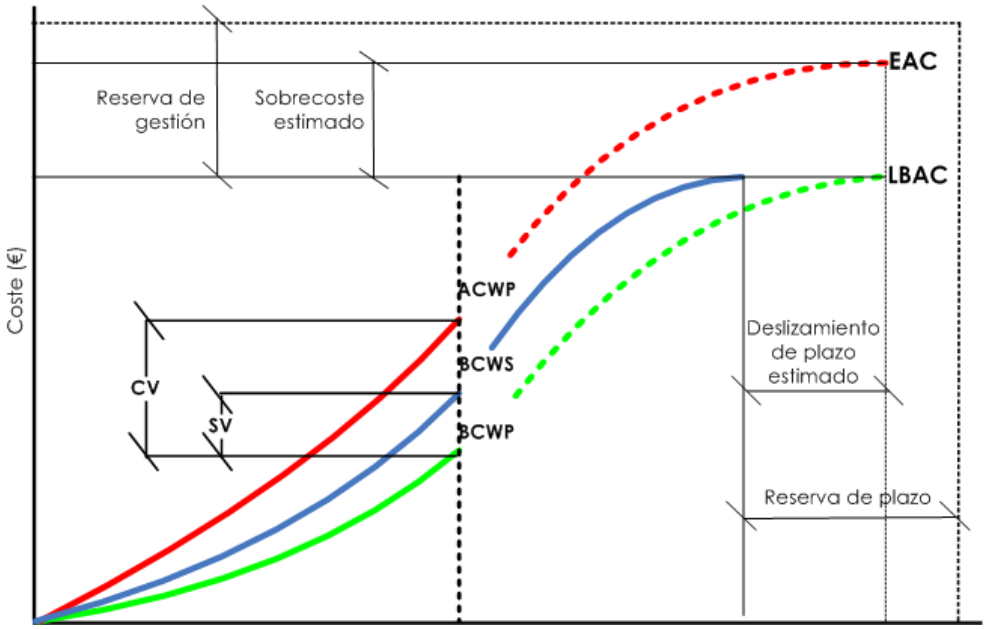

Figura 20: Representación gráfica de EVS.

rápidamente analizar el estado del proyecto. A continuación se describen los indicadores más importantes,

- Varianzas entre las tres variables indican si el proyecto va o no retrasado y si hay o no sobrecoste.
  - Varianza de coste (CV – *Cost Variance*) es la diferencia entre el valor conseguido y el valor actual, $CV = BCWP - ACWP$. Si el valor de CV es positivo es que se trabaja con más eficiencia que la planificada, y al contrario si es negativo.
  - Varianza de plazo (SV – *Schedule Variance*) es la diferencia entre el valor conseguido y el valor presupuestado, $SV = BCWP - BCWS$. Si la diferencia es positiva, el proyecto va adelantado, pero si por el contrario es negativa, el proyecto se está retrasado.
  - Varianza de coste relativa, $CVP = (BCWP - ACWP)/ACWP$.
  - Varianza de plazo relativa, $SVP = (BCWP - BCWS)/BCWS$.

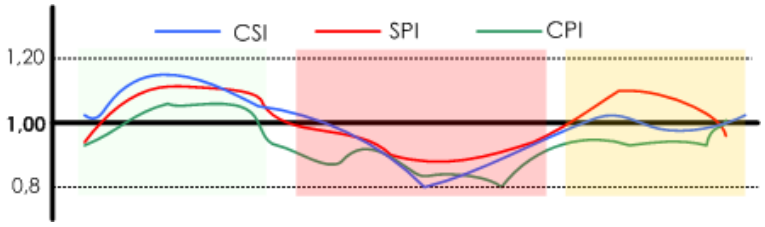

Figura 21: Análisis de los índices de EVS

- Índices. En los indices, valores cercanos a 1 indican que la ejecución tanto en costes como en plazos está muy cerca de la planificada. Cuando más se alejan de 1 por abajo, es que el proyecto se está retrasando o costando más de lo planificado, pero si están por encima de 1, el proyecto se está realizando mejor que lo planificado. Los índices tienen la ventaja que permiten comparar proyectos, sin tener en cuenta su tamaño. La figura 21 muestra el uso de los índices gráficamente, en la parte izquierda, el proyecto va mejor de lo planificado, mientras que en el centro va mucho peor trabajando más y consiguiendo menos de lo planificado

  - Índice de eficiencia de coste (CPI – *Cost Performance Index*): $CPI = BCWP/ACWP$.
  - Índice de eficiencia de plazo (SPI – *Schedule Performance Index*): $SPI = BCWP/BCWS$.
  - Índice de coste y plazo (CSI – *Cost-Schedule Index*): $CSI = CPI \cdot SPI$.
  - Índice de Eficacia para la Conclusión (TCPI – *To Complete Performance Index*) es el cociente del trabajo restante y el presupuesto restante:
    $TCPI = (LBAC - BCWP)/EAC - ACWP)$
    Es equivalente a CSI.

- Previsiones. Son indicadores que nos ayudan a determinar el estado al final del proyecto, es decir, su progreso. Generalmente se asume que los

índices de eficiencia de costes y plazos se mantendrán constantes.

- Valor estimado hasta la conclusión (ETC – *Estimated To Completion*) es la estimación del coste necesario para la finalización del proyecto. Al final del proyecto será cero ya que el coste estimado será igual que el coste real. $ETC = EAC - ACWP$
- Valor estimado a la conclusión (EAC – *Estimated At Completion*) se calcula extrapolando las curvas del valor realizado real ($ACWP$) y el conseguido ($BCWP$):

  $EAC = ACWP + \frac{(LBAC - BCWP)}{CPI}$

  Otra forma de calcular EAC es:
  $EAC = LBAC/CPI$

- Constantes del proyecto en relación a los costes y plazos.
  - Valor presupuestado del proyecto (LBAC – *Line Base at Completion*) es simplemente la estimación del coste del proyecto.
  - Reserva de gestión (MR – *Management Reserve*) es un sobrecoste permitido que el proyecto que puede ser usado de colchón para cualquier tipo de improvisto.
  - Reserva de plazo (SR – *Slippage Reserve*) es el tiempo que permitimos que el proyecto se deslize para cualquier tipo de improvisto que pueda surgir.
  - Ratio de coste, $CR = (LBAC + MR)/LBAC$.
  - Ratio de plazo, $SR = (LBAC + SR)/LBAC$.

A continuación se muestra el uso de estas variables e indicadores con un caso de estudio en un proyecto.

					Acumulado			
	H1	*H2*	*H3*	*H4*	*H1*	*H2*	*H3*	*H4*
BCWS	127	152	125	100	127	279	404	504
BCWP	115	120	130	139	115	235	365	504
ACWP	96	160	166	112	96	256	422	534
CV	19,00	-40,00	-36,00	27,00	19,00	-21,00	-57,00	-30,00
SV	-12,00	-32,00	5,00	39,00	-12,00	-44,00	-39,00	0,00
CVP	0,20	-0,25	-0,22	0,24	0,20	-0,08	-0,14	-0,06
SVP	-0,09	-0,21	0,04	0,39	-0,09	-0,16	-0,10	0,00
CPI	1,20	0,75	0,78	1,24	1,20	0,92	0,86	0,94
SPI	0,91	0,79	1,04	1,39	0,91	0,84	0,90	1,00
CSI	1,08	0,59	0,81	1,73	1,08	0,77	0,78	0,94
% completado	22,82	23,81	25,79	27,58	22,82	46,63	72,42	100,00
% gastado	19,05	31,75	32,94	22,22	19,05	50,79	83,73	105,95
EAC	12.622	20.160	19.307	12.183	12.622	16.471	17.481	16.020

3.4.1. Ejemplo EVS

Imaginemos un proyecto pequeño en el que marcamos cuatro fechas de revisión, podemos definirlas como *hitos*, en las que analizaremos el estado de proyecto tal y como muestra la figura 22 y con los siguientes parámetros:

- LBAC: 15.120 euros
- Fecha conclusión: 504 horas
- Reserva de gestión: 1.500 euros
- Reserva plazo: 60 horas
- Coste hora: 30 euros
- Ratio coste RC: 1,10
- Ratio plazo RS: 1,12

Tanto la gráfica del valor conseguido de la figura 23 como la de los índices de la figura 24 muestran que en general es un proyecto irregular en el que se cruzan las líneas de los tres valores y se producen desviaciones importantes en relación a lo planificado. Por ejemplo, podemos imaginar que es

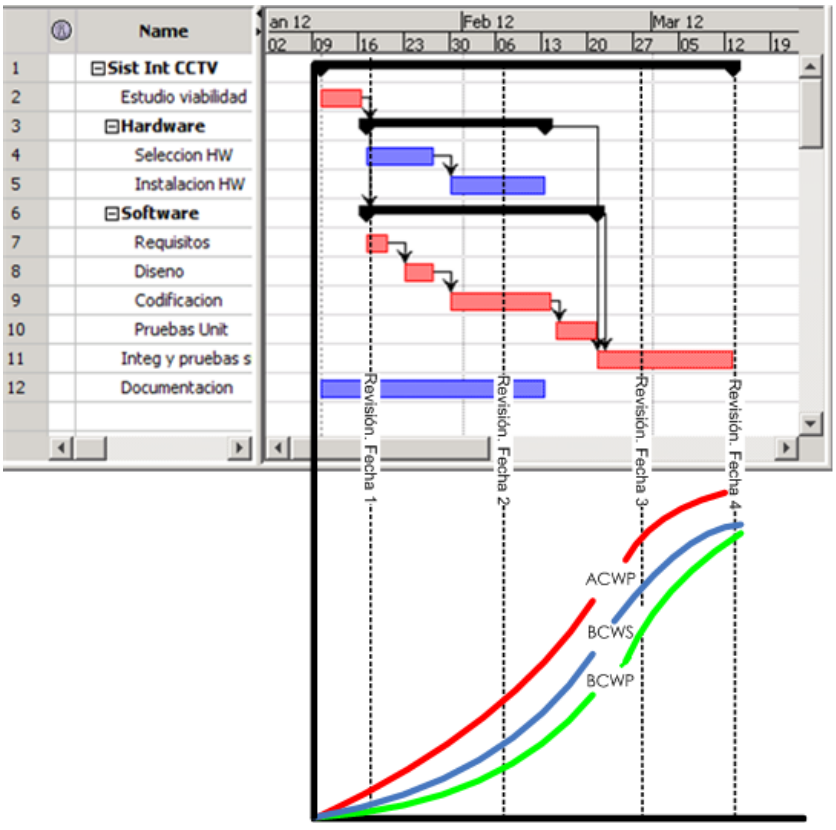

Figura 22: Fechas o hitos de análisis del proyecto

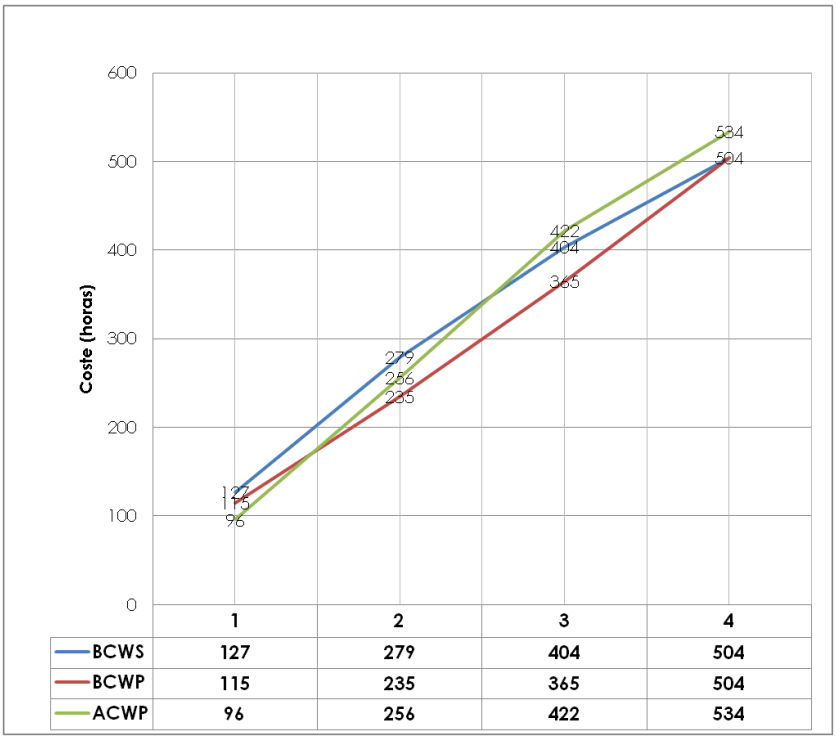

Figura 23: Gráfica de EVS

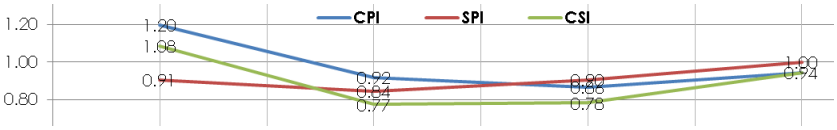

Figura 24: Índices del proyecto

un proyecto en el que se utiliza tecnología en la que la organización no tiene experiencia y/o personal novel. pero se puede apreciar que los gestores del proyecto han realizado acciones correctivas, terminado el proyecto con un pequeño sobrecoste pero dentro de la zona de reserva. Analizando las fechas en más detalle:

Fecha revisión 1 La varianza de coste es positiva por lo que la eficacia con la que se trabaja en este punto es positiva debido a que esta por debajo de lo que se invierte. En cuanto a la varianza de plazo, se observa un valor inicial negativo lo que predice una holgura en en plazo. Mirando los índices de eficiencia el estado del proyecto, los gestores del proyecto deberían incrementar las horas extra o las personas asignadas para mejorar su evolución temporal y acercar el SPI a uno (actualmente en 0,91), pero general el proyecto evoluciona correctamente al tener un índice de CPI superior a uno y la desviación de SPI es aun asumible.

Fecha revisión 2 En esta revisión el proyecto va francamente mal, ya que se trabaja más de lo esperado y con menor eficiencia de la prevista y con una proyección muy poco alentadora. Tanto la varianza de plazo como de coste son negativas decrecentándose en un gran porcentaje. Además, los índices de eficiencia, están por muy debajo de 1 y prácticamente fuera de una posible rectificación. Los gestores del proyecto deberían de bien renegociar el coste y plazos de entrega, bien incrementar las horas horas extra y el número de personas asignadas a este proyecto o ambas para subsanar estas desviaciones ya que el progreso de desarrollo se ha estancado encontrándose la falta de experiencia o motivación del personal.

Fecha revisión 3 El proyecto aun va mal, pero se ha mejorado. En este punto se ha conseguido más trabajo que lo planificado, pero a su vez ha costado más que lo que debería (por ejemplo, gracias a muchas horas extra pagadas a un coste superior que la hora normal), por ello la varianza de plazo es positiva, pero aun todos los valores acumulados siguen siendo negativos. Al igual que el la revisión anterior sigue siendo

necesaria es necesaria la mejora de la productividad. Además de horas extra o personal se podría considerar si es posible la sustitución de personal novel por otro más experimentado.

Fecha revisión 4 En el tramo final, el proyecto se ha recuperado y ha terminado a tiempo, ya que la varianza de plazo es cero. Parece que las medidas correctoras de los gestores de proyecto han dado sus frutos. Además, los índices de eficiencia han sido muy positivos y el único punto negativo es que ha costado más de lo planificado, pero este coste es inferior a la reserva de gestión, un 6 % frente a una reserva del aproximadamente el 10 % (16.020 euros frente a 15.120 euros planificados).

Los valores de EVS pueden fácilmente realizarse en hojas de calculo, pero las herramientas de gestión de proyectos como Microsoft Project[2] u Open Project[3] incluyen EVS como técnica de seguimiento de proyectos permitiendo definir distintos costes por persona, formas de distribuir el trabajo de una actividad (p.e., pico de trabajo al principio o al final, constante, forma de campana, etc.) y suelen incluir funcionalidad para su visualización. Por ejemplo, la figura 25 muestra la variable ACWP para una actividad y tiempo concretos en la realización del proyecto.

4. Revisiones y cierre del proyecto

Las revisiones e inspecciones son una técnica que en el caso de la gestión de proyectos nos ayudan a confirmar si se está cumpliendo con el plan definido, el proyecto cubre la funcionalidad requerida o la calidad de los productos y artefactos tanto intermedios como finales definidos en el proyecto. Las inspecciones por ejemplo dan visibilidad al avance del proyecto permitiendo dar por concluidos hitos o artefactos cuando se considera que tienen la calidad suficiente.

[2] http://office.microsoft.com/project
[3] `http://openproj.org/openproj`

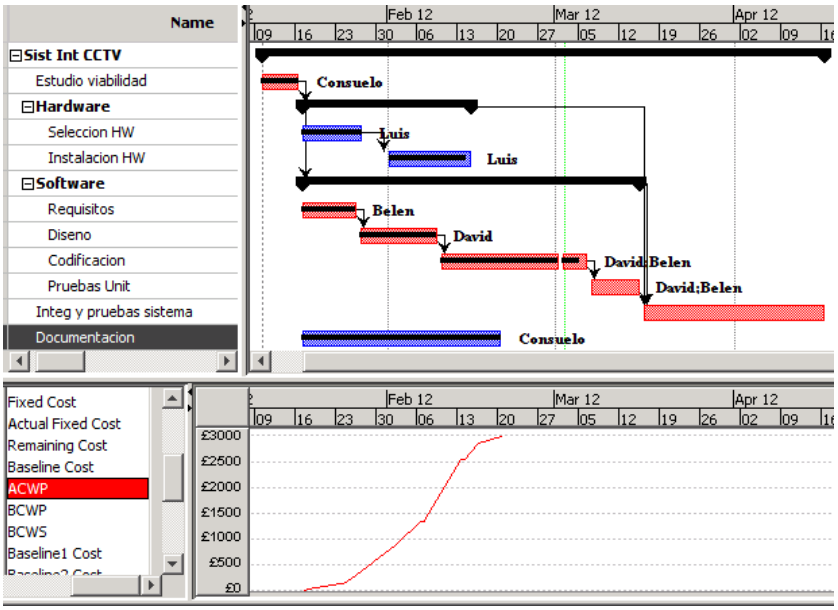

Figura 25: Ejemplo de cálculo ACWP con una herramienta de gestión de proyectos (Open Project).

La guía del cuerpo de conocimiento de gestión de proyectos (PMBOK) considera los siguientes tipos de revisiones:

- Revisiones del cronograma. Consisten en actualizaciones a fechas de hitos del proyecto aprobado, como respuesta a solicitudes de cambio por parte del cliente al alcance del proyecto o cambios en las estimaciones.
- Revisiones de rendimiento. Analizan actividades con variaciones en los constes en relación a los costes planificados, utilizando técnicas como la del valor conseguido como hemos visto en la sección 3.4.
- Revisiones en la identificación de riesgos, como veremos en la sección 6
- Revisiones de documentación. Consisten en revisiones formales de la calidad de todo tipo de documentación del proyecto.

Además es necesario realizar un análisis llamado *post-mortem*, donde se estudian mediante revisiones los problemas surgidos y futuras oportunidades

abiertas por el proyecto terminado. Una vez cerrado el proyecto hay que realizar un resumen de lo acontecido en todos sus aspectos, tanto tecnológicos como de gestión para mejorar futuros proyectos. Entre las actividades a realizar es la inclusión de toda la información posible en el histórico de proyectos, como hemos visto, esta información será usada para realizar estimaciones de futuros proyectos.

5. Gestión de recursos humanos

Para la correcta organización del personal dentro de un proyecto, los gestores de proyectos no sólo necesitan conocer el número aproximado de personas que necesitan para llevar a cabo el proyecto, sino también sus características personal para su asignación a las tareas que mejor se le ajustan. Entre estas características podemos citar, su habilidad e interés para llevar a cabo cierto tipo de trabajo, experiencia, conocimiento de procesos y herramientas, etc. También es necesario conocer sus habilidades personales de gestión, comunicación con el resto del equipo y nivel de responsabilidad. A las personas, se les suele clasificar como *extrovertidos*, los que tienden a decir lo que opinan, e *introvertidos*, que tienden a preguntar opiniones antes de llevar a cabo ciertas tareas. Por otro lado, también se puede clasificar a las personas por *intuitivos*, que se dejan llevar o se fían de los sentimientos e intuición, y *racionales*, aquellos que buscan más la lógica y los hechos antes de tomar decisiones. La combinación de estas cuatro tipos de personalidad además de las características del personal descritas, pueden indicar al gestor de proyectos qué individuos son los más apropiados a cada tipo de tareas.

Es muy importante tener en cuenta la productividad individual del personal. La productividad implica efectividad y eficiencia tanto a nivel individual como organizativo. Por efectividad se entiende el cumplimiento de los objetivos, y por eficiencia el cumplimiento de los objetivos con la menor cantidad de recursos.

Es conocido que en informática hay personas mucho más productivas

que otras, los estudios atribuyendo la varianzas más bajas la sitúan que hay personas hasta 5 veces más productivas que otras teniendo similar rango y salario. Otros estudios consideran que dichas diferencias pueden llegar a ser de 100 a 1. De hecho, G. Gordon Schulmeryer [] describió el "programador de productividad negativa neta", el cual dice que en casi todos los proyectos hay programadores de productividad negativa, los cuales causan más efectos negativos que positivos. El autor destaca que en un equipo de 10, puede haber hasta 3 personas que introducen tal tasa de errores que los convierten su productividad neta en negativa.

Otro aspecto a tener en cuenta es la organización del personal dentro de la organización, generalmente se consideran dos tipos de equipos: *programador jefe*, que es una estructura jerárquica donde el mejor miembro del equipo lo dirige asignando tareas y la aproximación no egoísta, donde es un modelo más matricial donde la responsabilidad está más repartida entre todos los miembros del equipo.

6. Gestión y análisis del riesgo

Riesgo en la ingeniería del software se define como la posibilidad de que ocurra un evento no deseado que afecte a los objetivos de plazo, coste, calidad del proyecto e incluso a supervivencia de una organización. El riesgo está relacionado con la incertidumbre, desconocimos si el evento ocurrirá o no, ya que en caso contrario se trataría de un hecho.

Definición [12]
Riesgo se define como la posibilidad de un evento, peligro, amenaza o que ocurra una situación situación.

Generalmente se definen dos estrategias en la gestión del riesgo. La estrategia *reactiva* consiste en adoptar las medidas necesarias una vez que evento ha sucedido, se suele utilizar el símil de esta estrategia consiste en ir apagan-

do fuegos. Esta estrategia es por supuesto, no recomendada. Por otro lado, la estrategia *proactiva* consiste en prevenir los riesgos, siguiendo un proceso para prevenir o mitigar los problemas que puedan acontecer, identificándolos, analizando su probabilidad e impacto, teniendo planes de contingencia para tomar las medidas necesarias y continuamente controlando el proceso de desarrollo. Estándares como el SWEBOK establece las siguientes actividades del proceso de gestión de riesgos:

- Identificación y análisis de riesgos, en términos de identificar qué puede ir mal, cómo, por qué y cuales serían las posibles consecuencias.
- Evaluación de riesgos críticos, para situar los riesgos más significativos en cuanto a exposición y perspectiva de pérdida.
- Plan de mitigación de riesgos, que contendrá la estrategia para su elusión o minimización de su impacto.

Identificación del riesgos. La identificación consiste en identificar que riesgos se van a controlar. Existen distintas estrategias como tener listas donde se comprueban si hay riesgos aplicables al proyecto, descomponer el problema y analizarlo, etc. Es importante identificar todos los riesgos potenciales ya que de otra forma no serían considerados en etapas posteriores. Generalmente, los riesgos se clasifican en tres categorías distintas:

- Riesgos relacionados con el proyecto que afectan a los plazos o recursos.
- Riesgos relacionados con el producto que afectan a la calidad del software que se esta desarrollando.
- Riesgos relacionados con el negocio, son riesgo que afectan a la organización, por ejemplo si se esta llevando a cabo un proyecto que si fracasa podría tener consecuencias graves para la organización (acciones legales, reducciones de plantilla, etc.).

Después el en análisis de riesgos se evaluar su probabilidad y la gravedad de los mismos teniendo en cuenta las siguientes definiciones:

Cuadro 14: Ejemplo Riesgos

Orden	*Riesgo*	*Probabilidad*	*Impacto*
1	Cambios en los requisitos una vez iniciado el proyecto	Alto	Serio
2	No poder contratar el personal necesario	Alto	Serio
3	El tamaño del software se ha infravalorado	Alto	Tolerable
4	Disponibilidad de cliente menor de la esperada	Moderado	Tolerable
5	Problemas con las licencias de las herramientas	Moderado	Tolerable
6	Incendio oficinas	Muy bajo	Catastrófico
...	....	...	

- El *impacto del riesgo* es la perdida asociada con el riesgo. Generalmente se clasifican como insignificante, tolerable, serio o catastrófico. Muchas veces se considera el coste.
- La probabilidad del riesgo, se suele expresar en rango discretos: muy bajos (<10 %), bajo (10-25 %), moderado (25-50 %), alto (50-75 %) o muy algo (75-100 %).
- La exposición al riesgo se calcula mediante el impacto del riesgo y su probabilidad.

Generalmente los riesgos se ordenan en una tabla (como por ejemplo la tabla 14), que según avanza el proyecto se debe de ir actualizando según pasa el proyecto ya que su impacto o probabilidad pueden cambiar.

Además, puede hacerse un análisis cuantitativo, por ejemplo la figura 26 muestra como la probabilidad de que un evento ocurra, se puede multiplicar por su coste en cada rama del árbol. En la figura, podemos observar que el riesgo combinado de no realizar inspecciones (de código o diseño) supera al de no realizarlas, por lo que es posiblemente más seguro realizarlas. Además si este tipo de análisis cuantitativo puede utilizarse para priorizar riesgos y tener más controlados aquellos con más prioridad.

Figura 26: Ejemplo de cálculo cuantitativo de exposición de riesgos

En la planificación de riesgos se considera cada riesgo y su estrategia para gestionarlo. Dependiendo del riesgo y su efecto, se pueden utilizar estrategias que lo eviten, por ejemplo, si la automatización de un módulo es demasiado compleja, puede hacerse manualmente hasta que mejore la tecnología. Otro tipo de estrategia consiste en la minimización del riesgo, por ejemplo, añadiendo personal redundante si hay muchas probabilidades de una alta tasa de abandonos. En la planificación de riesgos es necesario detallar los **planes de contingencia**, en las que se detalla que hacer con el cuando el evento adverso ocurre. Finalmente, hay ciertos tipos de riesgos que pueden desviarse a otras fuentes. Por ejemplo, comprando seguros que eviten pérdidas económicas.

7. Resumen

Hemos visto los conceptos básicos de gestión de proyectos: iniciación, planificación, seguimiento y cierre de proyectos.

En la planificación consideramos la estimación de esfuerzo y plazos, con diferentes técnicas, como se puede evaluar la bondad de dichas técnicas y la necesidad de calibrado, i.e., adaptación de modelos de regresión e inteligencia artificial a la situación actual de una organización. Además hemos estudiado como una vez definidas y organizadas las tareas, generalmente utilizando la descomposición de paquetes (WBS), los modelos de red (CPM y PERT) y

diagramas de Gannt pueden ser utilizados en la planificación y control del proyecto.

Para el seguimiento de proyectos, hemos estudiado la técnica del valor conseguido que permite combinar costes y plazos analizando tanto el estado actual de un proyecto como sus estimaciones futuras acorde a su estado actual. Además toda esta información se puede resumir en una sola gráfica, por lo que es una técnica de gran aceptación por los altos gestores en las organizaciones.

Otras actividades necesarias para la gestión de proyectos y que hemos visto, son la gestión de personal y la gestión de riesgos. La gestión de riesgos es una actividad fundamental en todo proyecto para evitar o mitigar situaciones adversas al proyecto que pueden afectar sus plazos o calidad, y a su vez a la organización en caso de proyectos vitales para las mismas.

8. Notas bibliográficas

Entre los estándares internacionales para la gestión de proyectos, cabe resaltar:

- PMBOK (A Guide to the Project Management Body of Knowledge). Modelo del PMI para la gestión de proyectos incluye la gestión da integración y alcance, plazos, costes, calidad, recursos humanos, comunicaciones, riesgos, adquisición e iteraciones entre procesos.
- El estándar de planes para la gestión de proyectos, ANSI/IEEE Std. 1058, Standard for Software Project Management Plans.
- El estándar de métricas de productividad, IEEE Std. 1045, Software Productivity Metrics.

También existe multitud de libros exclusivamente relacionados con la gestión en la ingeniería del software. Entre los más completos se encuentran el de R. Futrell et al [9].

Existe abundante bibliografía respecto a la estimación de proyectos software, pero se pueden recomendar los libros de Fenton y Plfleeger [7] y Dolado y Fernandez [6] en castellano. Otro referencia interesante, relacionada desde el punto de vista de la gestión de la calidad pero incluye los procesos de medición es el de S. Kan [13]. Otro libro recomendable y fácil de leer para la aplicación de modelos estadísticos en la ingeniería del software es el de K. Maxwell [14]. En cuanto a COCOMO, las referencias original de COCOMO 81 es la de Bohem [4] y para COCOMO II, [5]. Para los puntos de función, la referencia original es la Albretch [2], más accesible la referencia de Albrecht y Gaffney [3], y como libro el de Garmus y Herron [10]. El trabajo seminal para los sistemas dinámicos es el de J. Forrester [8], en la ingeniería de la referencia fundamental es la de Abdel-Hamid [1].

En relación a repositorios de datos sobre gestión existen varias:

- FLOSS (*Free/libre Open Source Software*) Metrics . El objetivo de este repositorio es disponer de una gran base de datos con información y métricas de cientos de proyectos software *open source*.
  `http://www.flossmetrics.org/`

- PROMISE (*Predictor Models in Software Engineering*) es un repositorio contiene múltiples bases de datos clasificadas según su propósito (estimación de defectos, estimación de costes, etc.) junto con los artículos que las han usado. Esta base de datos está relacionada con la conferencia que lleva el mismo nombre.
  `http://promisedata.org/`

- ISBSG (*International Software Benchmarking Standards Group*) es una base de datos clásica, de pago. La experiencia nos ha demostrado que es muy difícil aplicar algoritmos debido a la heterogeneidad de los datos y a los diversos enfoques de *puntos de función* utilizados, lo que dificulta en lo referente a inferencia sobre los datos.
  `http://www.isbsg.org/`

- UDD (*Ultimate Debian Database*). Esta base de datos recoge informa-

ción sobre todos los aspectos de desarrollo de Debian, incluyendo los fuentes de Ubuntu, defectos, migraciones, etc.
`http://udd.debian.org/`

La lista de eventos donde se pueden encontrar ponencias sobre este tema es muy extensa. Mencionamos a continuación algunos que específicamente están orientados al tema

- PROMISE, conferencia dedicada exclusivamente a los modelos de predicción y análisis de datos como se comentado anteriormente.
- MSR (Mining Software Repositories). El mismo título de la conferencia indica que abarca la minería sobre cualquier tipo de datos relacionados con el software. (`http://www.msrconf.org`).
- SCAM (*Source Code Analysis and Manipulation*) aunque esta conferencia se centra en la manipulación de código (p.e. con técnicas como *slicing*), también se incluyen entre los tópicos de la conferncia la minería de datos (`http://ieee-scam.org/`).

ESEM (*Empirical Software Engineering*). Esta conferencia se centra en los estudios empíricos y métricas del software (anteriormente se denominaba *Software Metrics*) pero también se presentan artículos de minería de datos aplicados a proyectos software.

Existen un buen conjunto de revistas internacionales donde se publican resultados de minería de datos en Ingeniería de Software, como son Empirical Software Engineering Journal, Journal of Systems and Software, Information and Software Technology, IEEE Trans. on Software Engineering.

Con respecto a la gestión del riesgo, destacamos el libro de Hall [11] o McManus [15].

Referencias

[1] T. K. Abdel-Hamid. *Software Project Dynamics: an integrated approach*. Prentice-Hall, 1991.

[2] A. J. Albrecht. Measuring application development productivity. In *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium*, pages 83—92. IBM Corporation, October 1979.

[3] A. J. Albrecht and J. E.Gaffney. Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering*, 9(6):639–648, 1983.

[4] Barry W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[5] Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[6] José Javier Dolado and Luis Fernandez. *Medición para la gestión en la ingeniería del software*. Ra-ma, 2000.

[7] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: a Rigorous & Practical Approach*. International Thompson Press, 1997.

[8] J.W. Forrester. *Industrial Dynamics*. The MIT Press, Cambridge, Mass., 1961.

[9] Robert T. Futrell, Donald F. Shafer, and Linda Shafer. *Quality Software Project Management*. Prentice Hall PTR, 2002.

[10] David Garmus and David Herron. *Function Point Analysis: Measurement Practices for Successful Software Projects*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[11] Elaine M. Hall. *Managing risk: methods for software systems development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[12] IEEE. IEEE standard glossary of software engineering terminology, 1990.

[13] Stephen H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edt. edition, 2002.

[14] Katrina D. Maxwell. *Applied Statistics for Software Managers*. Prentice Hall PTR, 2002.

[15] John McManus. *Risk Management in Software Development Projects*. Butterworth-Heinemann, Newton, MA, USA, 2003.