PUNTOS DE FUNCIÓN

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

De los beneficios del método, Steve McConnell se hace eco en su libro «*Estimación de software: desmitificando este arte oscuro*» (McConnell, 2006). El autor comparó los datos de un conjunto de estimaciones con los datos finales tomados al término de los proyectos, obteniendo como resultado que en un 33% de los casos en los que se empleó la técnica Delphi la estimación correcta estaba dentro del rango delimitado por los estimadores.

Estimación de expertos por analogía

Esta técnica consiste en realizar la estimación del proyecto actual comparando dicho proyecto con otros anteriores, de características y ámbito similares. Obviamente, se presupone que la organización mantiene una base de datos con proyectos realizados, o utiliza algún repositorio disponible. Uno de los más importantes es el mantenido por ISBSG (*International Software Benchmarking Standards Group*), un repositorio de métricas de gestión de proyectos de diferentes sectores (banca, telecomunicaciones, etc.), donde se pueden encontrar proyectos similares al que queremos estimar en relación al tamaño, lenguajes, tipo de base de datos, uso de herramientas CASE, etc., y consecuentemente, predecir el esfuerzo necesario total y por fases del ciclo de vida, la productividad, el tiempo de desarrollo, la tasa de defectos, etc.

10.4.2 Puntos de función

Los **puntos de función** –originalmente creados por Albretch a finales de los años 1970 en IBM– son una de las técnicas más usadas para la estimación de tamaño en proyectos software. Existen variantes a este modelo, siendo las más conocidas los puntos de función IFPUG y COSMIC.

De modo simplificado, y para entender el funcionamiento de los puntos de función en general, diremos que se basan en la cuenta de elementos para medir su funcionalidad. Por ejemplo, en IFPUG se hace uso del concepto de punto de función no ajustado (UPF, *Unadjusted Function Points*) entre los que se consideran:

- *Entradas externas* (entradas de usuario), por ejemplo, selecciones de menú.
- *Salidas externas* (información para el usuario), como mensajes o informes.
- *Consultas*: entradas interactivas que requieren una respuesta del sistema.
- *Ficheros externos*: son las interfaces con otros sistemas.
- *Ficheros internos*: cualquier entidad persistente manejada por el sistema.

Estos elementos se clasifican según su complejidad (simple, media o compleja) por un valor de ajuste acorde a ciertas tablas de ponderación (*ver* Tabla 10.1). Aunque existen guías

para clasificar cada elemento dentro de la tabla de complejidad, abordar dichas directrices se queda lamentablemente fuera del alcance de este libro.

Tabla 10.1: Factor de ponderación según complejidad

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

Como ejemplo de puntos de función, imaginemos en un sistema los elementos y la complejidad indicados en la Tabla 10.2.

Tabla 10.2: Ejemplo de puntos de función 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**

Los puntos de función no ajustados son la suma de la multiplicación del número de elementos por su peso:

$$PF_{NoAjustados} = \sum_{i=1}^{15} numElementos_i \cdot peso_i$$

Una vez calculados los puntos de función no ajustados, se procede a aplicar un factor de corrección, llamado *ajuste de complejidad técnica* (*PCA*, *Processing Complexity Adjustment*) que depende de los catorce atributos que se muestran en la Tabla 10.3. 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 de cada factor. A continuación, el valor de ajuste se calcula con la ecuación:

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

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

Tabla 10.3: 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 usada masivamente	11	Facilidad de operación
5	Tasa de transacción	12	Facilidad de instalación y conversión
6	Entrada de datos *on-line*	13	Puestos múltiples
7	Eficiencia para el usuario	14	Facilidad de cambio

variar un $\pm 35\%$ dependiendo de la complejidad y cada factor influye un 5%. Una vez calculados los puntos de función ajustados, se estiman las líneas de código (*LoC*) mediante tablas que muestran el número medio de líneas de código por punto de función al estilo de la Tabla 10.4. Finalmente, el esfuerzo se puede calcular con la productividad pasada de la organización.

Tabla 10.4: Lenguajes y número medio de líneas de código por punto de función

Lenguaje de programación	*Media LoC/PF*
Lenguaje ensamblador	320
C	128
Cobol	105
Fortran	105
Pascal	90
Ada	70
Java, C++	53
Generadores de código	15
Hojas de cálculo	6
Lenguajes gráficos	4

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:

$$PF_{ajustados} = PF_{noAjustados} \cdot PCA = 372$$

Después, basándonos en la Tabla 10.4, sabemos que aproximadamente 128 líneas de código en C equivalen a un *punto de función*, por lo que tenemos:

$$LoC = 128 \times PF = 47.616 \text{ líneas de código en C}$$

Puesto que el esfuerzo se puede calcular como el tamaño dividido por la productividad, asumiendo una productividad de 12 *PF/personas-mes* tendremos:

$$esfuerzo = PF/productividad = 372/12 = 58 \text{ personas-mes}$$

y finalmente, si la media es de 3.000 euros por persona-mes (250 euros por PF) el coste será de 290.000 euros, aproximadamente.

10.4.3 Modelos algorítmicos o paramétricos

Los modelos algorítmicos o paramétricos representan relaciones –en forma de ecuaciones– entre variables, generalmente el esfuerzo y el tamaño del software, pero pueden incluir otras características del proyecto como la complejidad o los factores relacionados con la organización del proyecto.

Existen multitud de modelos paramétricos, que se pueden agrupar en abiertos –públicos o de caja blanca, si se conocen las ecuaciones que los componen– o propietarios –cerrados o de caja negra, donde una institución comercializa 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, modelos que 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.

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 el tamaño, la complejidad, etc., utilizando algún tipo de curva de regresión. La Figura 10.2 muestra dos ejemplos de curvas de regresión, donde el esfuerzo (e) se relaciona con el tamaño (t). Hay muchos tipos diferentes de regresiones matemáticas como por ejemplo la 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.

COCOMO

El modelo paramétrico probablemente más conocido es el de COCOMO (*COnstructuctive COst MOdel*), dado que fue uno de los primeros modelos abiertos desarrollados. Fue propuesto por B. Boehm en 1981 basándose en 63 proyectos –principalmente de la NASA– y actualizado en el año 1995 bajo la denominación COCOMO II, que veremos más adelante. En el modelo inicial, también conocido como COCOMO 81, se distinguen tres tipos de proyectos según su dificultad y entorno de desarrollo: