Juan Grompone

Gestión de Proyectos de Software

1996

Versión 2.0, corresponde a la versión 3.1 de la planilla electrónica GPS.XLS. Es de libre circulación. 2004.

Primera Edición
Editorial La Flor del Itapebí.
Olmer S.A.
Luis Piera 1917, ap. 401
Montevideo, Uruguay.

Telefax: 49 01 91

ISBN 9974–592–05–4

CONTENIDO

1. CONCEPTOS GENERALES

1.1 Planteo del problema de la gestión

Este libro es el resumen de más de 20 años de trabajo práctico en la gestión de proyectos. No es un libro teórico. Persigue la finalidad de ilustrar, mediante estudio de casos, el empleo de herramientas prácticas. Con esta finalidad, al final de libro hay un disquete con una implementación de las funciones empleadas efectivamente en la gestión de proyectos.

La gestión de proyectos de software persigue la misma finalidad que todas las gestiones de proyectos en ingeniería:

- estimar qué sucederá con un proyecto nuevo
- analizar qué sucedió con un proyecto ya finalizado

En todos los casos se tratará de dar respuestas cuantitativas a preguntas precisas tales como:

- ¿Cuál será el plazo de entrega?
- ¿Cuántas personas necesito?
- ¿Cuánto costará el proyecto?

La gestión de proyectos de software es una rama especializada de la Ingeniería del Software. Posee, además de los métodos generales de la ingeniería, metodologías propias.

En este libro nos ocuparemos fundamentalmente de las metodologías propias de los proyectos de software. Las metodologías generales se supondrán conocidas y solamente se mencionarán cuando corresponda.

1.2 Gestión de proyectos de software

No debe perderse de vista nunca que lo que intentamos hacer **no es una rama de la brujería** sino una rama de la Ingeniería del Software. No se busca preparar gurús sino ingenieros, no se apela a oscuros mecanismos sino a datos numéricos en información medible.

No debe confundirse una **ecuación empírica**, obtenida del análisis de muchos casos y sometida reiteradas veces al contraste con la experiencia práctica con una forma de brujería. No debemos olvidar que toda ciencia fue, alguna vez, solamente una colección de ecuaciones empíricas.

Por esta razón, estamos hablando de una rama de la ingeniería que:

- Emplea metodologías bien definidas.
- Realiza medidas repetibles y confiables.
- Estima costos y tiempos.
- Da elementos para la gestión de los proyectos.
- Replantea resultados para ajustar la información disponible.

De esto se trata este libro: de metodologías. Como resultado de estas metodologías, el libro posee una planilla electrónica Excel, GPS (Gestión de Proyectos de Software), que implementa la mayoría de los métodos presentados.

Este libro tiene pasajes teóricos. En la mayoría de los casos conocer estos pasajes teóricos no es necesario para aplicar correctamente la metodología. En estos casos el libro advierte con este dibujo:

que el pasaje es eminentemente teórico y que **puede ser omitido** sin perjuicio para las aplicaciones prácticas.

Como las técnicas de gestión solamente quedan claras a través de ejemplos, existe una colección de estudio de casos. Muchas veces es en estos casos donde se encuentra la información más interesante. Es sumamente recomendable no saltear ninguno de ellos.

1.3 Diferentes tipos de proyectos

En proyectos de software debemos diferenciar tres tipos diferentes de proyectos, desde el punto de vista de su gestión:

- **Proyectos nuevos**: se busca analizar costos, tiempos y cantidad de personas. Es el caso más difícil de todos.

- **Replanteo de proyectos viejos**: se busca afinar las metodologías de estimación. Es la principal fuente de información.

- **Extensiones** o ampliaciones de un proyecto existente: Es un caso intermedio donde se desea tener buena precisión en plazos y costos.

Siempre se debe tener en cuenta que los tres casos presentan situaciones diferentes y no pueden ser tratados por igual.

1.4 El tamaño de los proyectos

Los proyectos de software son diferentes por la sola razón de su **tamaño**. Por el momento no podemos dar criterios de diferenciación cuantitativa, pero debemos tener presente que existen tres categorías diferenciadas de proyectos, con problemas diferentes cada una:

- Proyectos **pequeños:** consisten solamente en implementación. No tienen costos indirectos importantes.

- Proyectos **medianos**: es un caso intermedio entre los otros dos.

- Proyectos **grandes**: poseen implementación, pero hay más cosas. Poseen gerencia de proyecto, control de calidad, capacitación de

personal, hay un plan de mantenimiento, hay documentación importante para uso interno y externo. Se genera información para mercadeo.

Un error clásico en la historia de la gestión de proyectos fue no advertir la existencia de tres categorías diferentes. Todavía hoy se sigue pensando que la información o la experiencia adquirida en proyectos pequeños puede servir para proyectos medianos o grandes. Este es uno de los orígenes de los resultados catastróficos en la gestión de proyectos.

Desde un punto de vista cuantitativo, los proyectos se pueden diferenciar mediante valores límites –algo convencionales, es necesario agregar– que permiten una estimación más clara.

La constitución del equipo de trabajo es uno de los mejores indicadores del caso que se considera. Así por ejemplo, los proyectos pequeños poseen:

- Menos de un año de tiempo de desarrollo.
- Menos de 25 meses–persona de esfuerzo total.
- Menos de 3 personas en el equipo de trabajo.

Por el contrario, los proyectos grandes poseen:

- Más de 3 años de tiempo de desarrollo.
- Más de 100 meses persona.
- Más de 10 personas en el equipo de trabajo.

Como es interesante observar, esta clasificación no hace ninguna referencia a la tecnología empleada ni, por lo tanto, al tamaño del proyecto. Más adelante, cuando se introduzcan métricas de proyecto se podrá relacionar el tamaño del proyecto con estas medidas.

Un equipo de trabajo de menos de 3 personas puede trabajar con una documentación informal, un equipo de más de 10 personas, durante varios años, no puede confiar en los contactos personales ni en la memoria de la gente. Es más, puede ocurrir que muchos de los que comienzan el proyecto sean reemplazados por otros, en plazos tan largos. Esta diferencia en el

manejo interno de la información hace un gran distinción entre los dos extremos de tamaños de proyectos.

Es muy importante observar que la barrera de 3 años es una barrera dramática para la informática. En tres años hay cambios importantes en las plataformas. De acuerdo con cifras usualmente aceptadas (ley de Gordon Moore), la capacidad de almacenamiento y la velocidad de las plataformas **se multiplica por diez** en tres años. Tenemos un orden de magnitud de crecimiento. Esto implica, para un proyecto grande, que:

- La plataforma para la cual diseñamos será obsoleta en el momento de entrega.
- La disponibilidad de memoria, la velocidad de procesamiento y el tamaño del almacenamiento, que al comenzar el proyecto puede ser una limitación del diseño, puede dejar de serlo al final del proyecto. Este elemento no debe dejarse de tener en cuenta.
- Es posible que aparezcan nuevas tecnologías y nuevas herramientas de trabajo con las cuales no se contaba al comenzar el proyecto.
- Es posible que aparezcan nuevas exigencias con la cuales no se contaba al comenzar el proyecto.

Estas consideraciones muestran que hay diferencias profundas en el armado y realización de un proyecto grande. Este es uno de los temas centrales de la gestión de proyectos.

1.5 Breve historia de la gestión de proyectos

En la década del 50 no se tenía metodología. La computación era incipiente. No se diferenciaba bien entre el software y el hardware. Esta situación continuó hasta comenzada la siguiente década.

En la década del 60 ocurre la primera gran crisis de la gestión de proyectos de software: **la crisis del OS/360**. La nueva línea de computadoras de IBM, la línea **/360** planteó, por primera vez, la realización de un paquete de programación de tamaño **mediano**. Hasta este momento puede decirse que todos los proyectos eran pequeños o –si bien eran de tamaño mediano– se habían hecho en ambientes universitarios sin preocupase de plazos y costos.

Con el Sistema Operativo de **/360** IBM desborda todos los plazos y costos imaginables. De este primer gran atraso (que no fue el único atraso en la entrega del software, por cierto) nace la gestión de proyectos. [BRO72]

La década del 70 se caracteriza por la realización de los grandes estudios empíricos. La difusión de las computadoras y la aparición de las minicomputadoras hace que se disponga de muchos proyectos medianos y grandes. Con todo este material empírico se hacen gran cantidad de estudios. De estos estudios nacen modelos y metodologías. Estos son algunos de los resultados de este período:

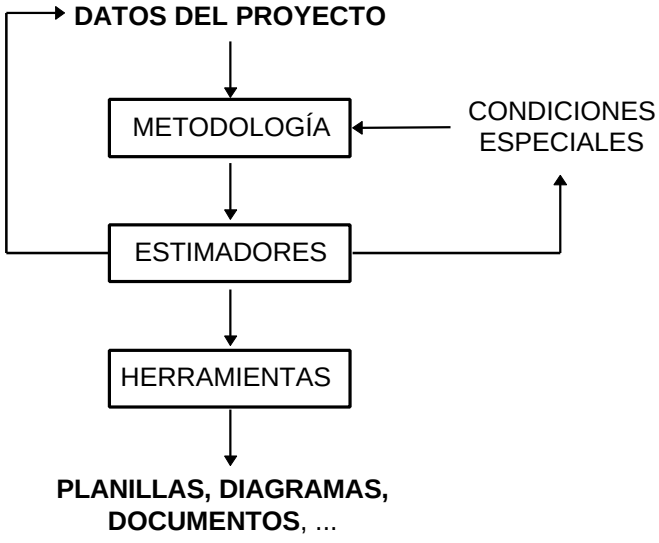
Farr – Zagorski, 1965
SDC, 1966
Aron, 1969
Naval Air Development Center, 1971
GRC, 1974
Tecolote Research Incorporated, 1974
Wolverton, 1974
ESD, 1975
RCA (Price–S) (< 1979)
Aerospace, 1977
Daly, 1977
Doty/RADC, 1977
Kustanowitz, 1977
Walston – Felix – (fin de los '70s)
Putnam, 1977
Albrecht, 1979–1984 (Function Points)
Boehm, 1980 (COCOMO)

La década del 80 es el período de confrontación de los modelos y metodología con los grandes proyectos de software. Es paradojal, pero la difusión de las computadoras personales hace que los proyectos sean más complejos y exigentes. De todas estas tecnologías sobreviven solamente unas pocas de la confrontación con la realidad. En este libro nos ocuparemos solamente de las últimas tres: Putnam, Albrecht y Boehm.

La década del 90, en la cual nos encontramos, es la época de normalización de las metodologías. Ya se ha separado lo útil de lo teórico, lo bueno de lo malo y se procede a normalizar las metodologías de modo que obtengan resultados consistentes y comparables.

1.6 Problemas básicos de la gestión de proyectos

En sus diferentes formas, todo problema de gestión de proyectos puede ser asimilado al siguiente diagrama:



Consideremos el caso de un proyecto nuevo. Existe un conjunto de datos del proyecto en cuestión. Estos datos, mediante una metodología, permite extraer consecuencias cuantitativas.

Es interesante diferenciar una metodología de un modelo. Un modelo es algo ambicioso: una teoría general sobre el proceso de diseño e implementación del software. En general, los modelos poseen muchas hipótesis y pocos casos de estudio. Por el contrario, una metodología es un conjunto de prácticas empíricas, contrastadas con la experiencia, de las cuales posiblemente no se tenga una explicación teórica. En este momento ha quedado establecido con bastante claridad que ninguno de los modelos ha servido de mucho y que solamente sobreviven las metodologías empíricas.

La metodología elegida permite, mediante estimadores empíricos, obtener resultados que se pueden contrastar con los datos del proyecto o con condiciones especiales que se desean imponer, tales como por ejemplo:

costo global, tiempo de entrega, calidad del producto, etc. Luego de algunas interacciones, se llega a una estimación adecuada.

Una vez completado el ciclo de interacciones, las herramientas tradicionales de gestión de proyectos –diagramas de Pert y Gantt, cálculo de costos, análisis de camino crítico, etc.– permiten que obtengamos todo el material necesario para la realización efectiva del proyecto:

- Costos, tiempos, cantidad de personas.
- Criterios para el seguimiento del proyecto: puntos claves, alertas tempranas de atraso o de costo alto.
- Criterios de medida para el replanteo.
- Criterios para la viabilidad económica.

En el caso del replanteo de un proyecto el diagrama es similar, excepto que posiblemente tengamos las incógnitas y los datos cambiados. En este caso el ciclo de interacciones permite ajustar cifras de modo de reconstruir lo acontecido a lo largo del proyecto.

1.7 Visión cuantitativa del proceso

El primer problema a definir consiste en determinar ¿cuáles son los datos de un proyecto? ¿De qué información debemos partir? La situación es diferente según sea un proyecto nuevo o un caso de replanteo. Consideremos primero el caso de un proyecto nuevo.

En un proyecto nuevo no hay nada hecho, la información que poseemos es **externa**, la visión que tiene alguien desde afuera, la visión que tiene el usuario. No sabemos nada acerca de objetos internos tales como cantidad de módulos, personas que participan o línea de código a utilizar. A lo sumo tenemos una cierta especificación del proyecto y algunas metas de costo y plazo de entrega que se deben alcanzar. Lo que sabemos suele ser muy poco, sin embargo este pobre material debería ser suficiente.

Lo que nos falta en un proyecto nuevo es la información de realización: costos, tiempos y personas.

Lo ideal sería disponer de una metodología que se basase en los datos externos del proyecto. Posterguemos por el momento la consideración de cuáles son estos datos. Siguiendo con esta ficción, lo ideal sería disponer de una **métrica** aplicada sobre los datos externos que nos midiera todo lo que hace falta. Luego, con estimadores, obtener los costos, tiempos y personas necesarios.

Con estos resultados, realizaríamos la comparación con las metas externas. Verificaríamos si el costo y el plazo de entrega es aceptable. Si no lo es, debemos replantear el proyecto, modificar algunos de sus datos externos si no hay ajuste con las metas y proceder nuevamente a recalcular. Así hasta llegar a una situación de acuerdo. Una vez logrado esto, basta con aplicar las herramientas clásicas de gestión para dividir el proyecto en tareas, tiempos y otros elementos que permiten ejecutarlo.

En el caso de replanteo de un proyecto la situación es opuesta. Tenemos buenos registros de cuanto costó el proyectos, en qué tiempo se hizo y cuantas personas trabajaron. Pero no hemos registrado nada acerca de los datos externos del proyecto, no lo hemos medido en lo previo.

El punto de partida consistiría ahora en recuperar los datos externos del proyecto. Para esto realizamos una estimación preliminar. Con esta estimación, aplicamos la metodología sobre los datos externos y luego estimamos costos, tiempos y personas. Estos elementos suelen estar registrados, de modo que se puede comparar los valores estimados con los datos registrados del proyecto. Si no hay ajuste, debemos replantear los supuestos datos externos hasta lograr un buen ajuste. Esta información es muy importante para la extracción de información para el futuro.

1.8 Los datos externos de un proyecto

Hemos aplicado con cierta libertad la idea de datos externos de un proyecto. Este es el momento de precisar esta idea.

En el caso de la construcción de edificios –el área más antigua y conocida de la ingeniería– se emplean continuamente datos externos de proyecto.

Si nos colocáramos en la situación de tener que diseñar y construir una vivienda, antes de comenzar deberíamos tener una idea acerca de lo que el usuario de la vivienda desea. Porque sabemos que existen una gran variedad de viviendas para una gran cantidad de usos. Tal vez lo primero que desearíamos saber es el tamaño de vivienda que desea el usuario, algo tan simple como los metros cuadrados que la vivienda posee, porque esta medida ya nos dará una buena idea de todo lo demás.

Es claro que si examinamos el problema con algo más de detalle, descubriremos rápidamente dos cosas que nos complican:

- Solamente el área edificada no alcanza. Dos construcciones de la misma superficie pueden diferir notablemente en todos sus detalles constructivos y, por lo tanto, en su costo.

- El futuro dueño de casa no tiene idea de los metros cuadros que necesita, no le importan, no sabe estimarlos ni calcularlos y si lo dice, posiblemente su dato sea erróneo.

Esto es claro, la superficie de una casa es un elemento técnico interno que al usuario no le interesa. Es algo así como la cantidad de módulo que quiere en su sistema. El usuario espera que sea el técnico quien le responda la pregunta y no al revés.

Por eso, de inmediato modificaremos nuestra estrategia y comenzaremos a solicitar verdaderos datos externos del proyecto. Por ejemplo, querremos saber ¿cuántos pisos debe tener la vivienda?, ¿cuántas habitaciones debe tener?, ¿cuántos baños?, ¿cuántas cocinas?, ¿qué tamaño de jardín? Todos estos elementos son **externos** y no **internos** al proyecto. El usuario que desea una casa puede responder claramente acerca de ellos.

Ahora sí, el proyectista comenzará a aplicar su metodología. Preparará una planilla con la información externa. Hará diversas preguntas hasta asegurarse que tiene un panorama bastante completo. Una vez que esté convencido de entender la complejidad o el tamaño del proyecto que enfrenta, realizará una estimación, habitación por habitación, en base a su

conocimiento previo, de la superficie que tiene cada una. Luego sumará y tendrá la respuesta que buscaba: la superficie total edificada de la vivienda.

Con esta información primaria, resultado de su metodología de trabajo, procederá ahora a investigar una información complementaria y preguntará, tal vez, acerca del lugar donde se construirá, del tipo de terminación y de todo otro elemento modificador que diferencia una vivienda de otra pero que incide finalmente en el costo.

Cuando todo este panorama esté claro aplicará un estimador de tipo empírico, basado en lo que ya sabe de la vivienda y en lo que cuesta este tipo de construcción y dará su estimación inicial de costo y plazo de entrega. Esta cifra será recibida por el usuario y, según sea, modificará las exigencias de su proyecto: quitará o agregará, mejorará o empeorará hasta llegar a un punto de acuerdo. Cuando este punto se logre, el proyectista tendrá un punto de partida firme para comenzar el proyecto de la vivienda. Es ahora que calculará ladrillos, cañerías y todo los elementos internos del proyecto.

Este panorama se puede trasladar con bastante exactitud a la ingeniería del software.

Al comenzar un proyecto de software –y, en general, siempre– es conveniente diferenciar la visión que tiene un implementador de la visión que tiene un usuario acerca del mismo proyecto. El implementador ve módulos, programas, algoritmos, registros bloqueados, recuperación de errores, lenguajes, procedimientos, y otros objetos técnicos. El usuario no ve nada de esto ni le importa, del mismo modo que no le interesa cuántos ladrillos se emplearán en construir su vivienda. Tanto le da que su sistema esté escrito en lenguajes de primera o de cuarta generación, tanto le da que haya manejador de base de datos como que no lo exista. No le interesa cuántos módulos hay ni cuales son los difíciles. Al usuario final solamente le interesan la **funcionalidad** del sistema (habitaciones, ventanas, jardines, en la vivienda).

La visión del proyecto que posee el **usuario** acerca de sus funcionalidades debería ser la información básica para la estimación de costos, plazos y dificultades, puesto que **no hay otra información**. Toda la información

interna es implementación, es una solución del problema, no el problema en sí.

Por esta razón, las metodologías de interés actual deben ser tales que se basen solamente en aquello que ve el usuario de su sistema informático:

- Datos de entrada al sistema: ingresos por pantalla y otras formas informáticas.
- Datos de salida del sistema: listados y otras formas informáticas.
- Datos almacenados en el sistema y la manera de consultarlos.
- Comunicación del sistema con otros sistemas informáticos.

La metodología ideal sería una **métrica algorítmica** sobre los datos externos que pudiese responder todas nuestras preguntas.

La respuesta clásica de la década del 70 fue crear **métricas empíricas** sobre algunos datos medibles del proyecto.

Lo que deseamos hoy es una respuesta de tipo ingeniería: una métrica **normalizada y repetible** que se aplica a datos objetivos y que permite reunir información empírica útil y confiable.

1.9 Los modelos de estimación

Los modelos de estimación son el resultado de dos elementos: **una idea**, a priori, acerca del proceso de fabricación del software y **una ecuación empírica** ajustada a partir de casos de estudio reales.

En la década del 70 todas las ideas se consideraban igualmente buenas. Hoy sabemos que algunas ideas conducen a ecuaciones empíricas erróneas.

Los modelos de estimación pueden se clasificados según diferentes criterios. Para el estudio que sigue será útil considerarlos, al menos, según tres puntos de vista diferentes.

Según el tipo de resultados que se obtienen, se pueden clasificar en:

- Modelos estáticos.
- Modelos dinámicos.

Los modelos estáticos dan estimaciones globales o promedio. Los modelos dinámicos, complementarios de los estáticos, permiten realizar estimaciones a lo largo del desarrollo del proyecto.

Según la cantidad de variables básicas que tienen en cuenta se pueden clasificar en:

- Modelos de una variable.
- Modelos de varias variables.

Las opciones abiertas significan un compromiso entre la sencillez (una variable) y la precisión (varias variables). Según las situaciones interesará más uno u otros.

Según el tipo de ecuaciones empíricas que se ajusten, se pueden clasificar en:

- Modelos lineales.
- Modelos potenciales o exponenciales.
- Otros modelos

Este punto será analizado con extensión más de una vez porque se toca aquí un punto muy delicado de la ingeniería del software.

2. MÉTRICAS TEÓRICAS DE PROYECTOS

2.1 Las métricas de proyectos

La idea que preside la gestión actual de proyectos es la convicción que existe una **métrica** –esto es, una manera de medir– para la complejidad de un proyecto. Esta medida nos permitirá responder a todas las preguntas acerca de la realización de un proyecto.

Regresemos al paralelo con una obra civil. Tal vez la métrica más importante sean los metros cuadrados edificados. La sola mención de la superficie edificada permite, de inmediato, tener una idea de tamaño y de allí, rápidamente una idea de costo, plazo de fabricación y todas los demás elementos que nos interesa saber. Es por esto que el área edificada es la métrica básica en una obra civil.

Pero en cuanto entramos en los detalles, será claro que debemos aplicar otras métricas: los metros cuadrados se deberán diferenciar según su uso y tipo de prestaciones. Emplearemos medidas para los metros de cañerías, de pavimentos o de terminaciones de las paredes, la cantidad de picos de luz y mil otros detalles más. No cabe duda que todos los elementos de una obra se miden, se planillan y terminan, finalmente, formando parte del análisis del proyecto.

La aspiración que se sigue con las métricas en la ingeniería del software es llegar a una precisión y claridad semejante.

2.2 La abstracción básica: la complejidad

En líneas generales la métrica básica de la programación es su **complejidad**. Cuanto más compleja sea, más difícil será de diseñar, de poner en operación, de quitarle sus errores, de documentar. El único problemas consiste en intentar medir esta supuesta complejidad.

La complejidad es algo difícil de definir y de precisar. En líneas generales es una métrica sobre los elementos de un proyecto informático. Es deseable que esta métrica:

- Tenga una definición simple y directa.
- Sea fácil de medir y sea coherente para diferentes jefes de proyecto.
- Tenga vinculación directa con los elementos básicos de los proyectos, especialmente con sus datos externos.
- Sea posible acumular evidencia empírica a lo largo del tiempo.

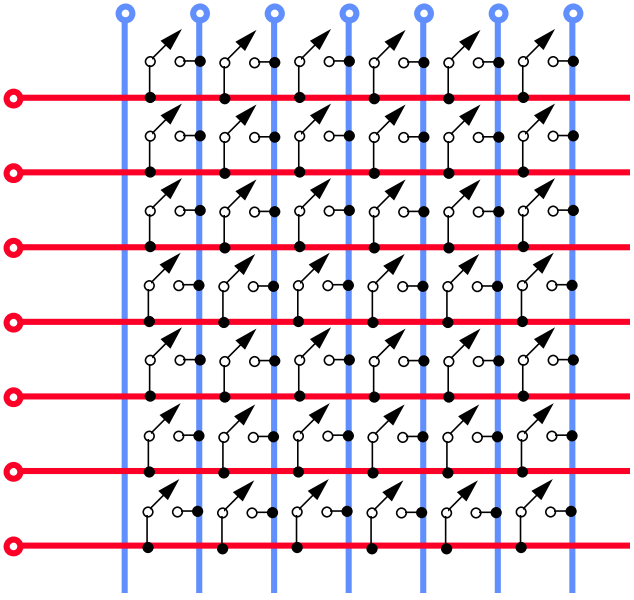
Hay numerosas propuestas de métricas, tanto teóricas como empíricas.

2.3 Bases teóricas de la complejidad

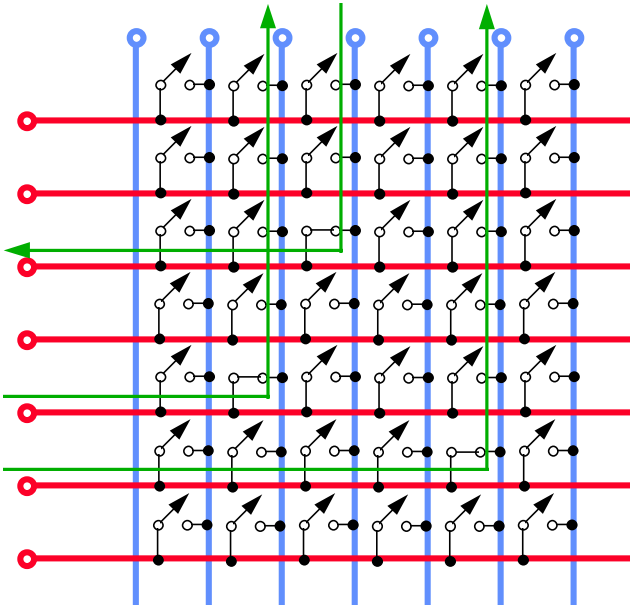
Complejidad de un sistema, planteado en general, es un tema que preocupó a los matemáticos hace una decenas de años. El resultado de todo esto fue la “teoría matemática de la complejidad”. A pesar de lo impresionante de su título, no se han obtenido resultados importantes.

En esta sección solamente intentaremos ilustrar el tipo de problemas que estudia esta teoría a través de un ejemplo simple y clásico: el conmutador telefónico. Supongamos una situación simple. Un conmutador telefónico debe atender las llamadas de ***n*** abonados y para esto tiene ***n*** líneas urbanas. En la solución tradicional, se dispondría de un cuadro de ***n*** *x* ***n*** crucen, en cada uno de los cuales existe la posibilidad de cerrar una llave.

La siguiente figura ilustra el diagrama de un conmutador elemental. Podemos imaginar que las barras verticales son los abonados y las horizontales las línea urbanas. En cada cruce hay una llave que puede cerrarse para establecer una comunicación. En este diagrama no hay ninguna comunicación establecida:



Si ahora imaginamos tres comunicaciones en operación, tendremos el siguiente camino de las comunicaciones y estado de las llaves.



La quinta línea urbana está conectada con el abonado tercero, la flecha indica que el origen de la llamada ha sido la línea urbana. La sexta línea urbana, de manera similar, ha llamado al abonado sexto. Finalmente, el abonado cuarto ha realizado una llamada que sale por la línea urbana tercera. En cada uno de los casos la llave correspondiente está cerrada en tanto que las líneas libres y las restantes llaves permanecen abiertas.

Para resolver este problema se ha acudido a un tablero que tiene n^2 llaves. Sin embargo, si se inspecciona el problema puede observarse que hay cantidad de llaves redundantes, que no se necesitan.

La teoría de la complejidad nació con la siguiente pregunta ¿cuál es el mínimo número de llaves que se necesitan para resolver el problema del conmutador telefónico?

Claude Shannon resolvió de una manera muy simple el problema del conmutador.

La primera llamada puede ocurrir en ***n*** caminos posibles.
La segunda llamada puede ocurrir en ***n–1*** caminos posibles.
....
El total de caminos posibles es el producto: ***n!***
El número de llaves mínimo, ***S***, cumple con la ecuación:

$$2^S \geq N!$$

Esta ecuación resuelve el problema de la cantidad mínima de llaves que necesita un conmutador telefónico. Aproximando ***n!*** Por su expresión asintótica:

$$S \geq N \log_2 N \quad (bits)$$

se tiene el número mínimo de llaves que se pueden emplear. En la medida que se involucra un logaritmo de base dos, la unidad de media empleada se debe llamar **bits**.

Esta ecuación se la toma como el primer resultado y la ecuación básica de la teoría matemática de la complejidad.

2.4 La métrica de Halstead

Halstead, hacia fines de la década del 70, propuso una métrica derivada de la teoría general de la complejidad y de algunas hipótesis adicionales acerca de la producción de la programación. Si bien sus ideas no han tenido aceptación práctica, son una referencia obligada para comprender las ideas que manejan otras teorías.

Para Halstead un programa es una sucesión de órdenes elementales. A su vez, cada orden posee operandos y operadores. Por cierto que es necesario precisar con cuidado qué cosa es un operando y qué es un operador; pero en esta presentación, que tiene solamente un carácter introductorio, no se entrará en estos temas. Sea:

n_1	número de operadores diferentes de un programa
n_2	número de operandos diferentes de un programa
N_1	número total de operadores que aparecen en el programa
N_2	número total de operandos que aparecen en el programa

Con estos elementos podremos introducir las sucesivas definiciones y propuestas de Halstead. Halstead llama **vocabulario de un programa** a la expresión:

$$n = n_1 + n_2$$

Llama **largo de un programa** a:

$$N = N_1 + N_2$$

Si suponemos que los operadores y los operandos alternan entre sí en un programa, que no existen trozos de programa repetidos (porque de existir, las reglas de la buena estructuración indican que se escribiría un subprograma) y algunas hipótesis más, entonces se tiene, que la cantidad de programas “bien estructurados” es:

$$n_1^{n_1} . n_2^{n_2}$$

Luego, el largo teórico, en bits, de un programa "bien estructurado" es la cantidad de bits que lo identifica, o sea:

$$n_1 . \log_2 n_1 + n_2 . \log_2 n_2$$

Esta expresión se asocia con el "contenido de información" de Shannon, con la "complejidad matemática" de algunos problemas y con otras ecuaciones similares. Por esta razón –más que por la presunta justificación de Halstead– esta métrica ha sido muy exitosa.

Halstead introduce muchos otros conceptos, basados en una visión "psicológica" del proceso de elaboración de la programación. Uno de ellos es el "volumen" del programa. Según su punto de vista, puesto que cada elemento de programa, operando u operador, contiene una información en bits

$$\log_2 (n_1 + n_2)$$

y hay N elementos para considerar, define el "**volumen**" (cantidad de elecciones que es necesario realizar para escribir el programa!) como:

$$V = N . \log_2 (n_1 + n_2)$$

También intenta definir la "**dificultad**" de un programa. Su métrica tiene que ver con el manejo de los operadores y operandos. El uso promedio de cada operador es N_2/n_2. Por otra parte, la operación más simple es invocar una función, lo cual requiere dos operandos. De modo que $n_1/2$ mide cuántas veces más "difícil" es el programa respecto a invocar una función. En total, la "dificultad" global será:

$$D = \frac{n_1}{2} \frac{N_2}{n_2}$$

A partir de esta ecuación, Halstead propone su ecuación de “esfuerzo”:

$$K = D \,.\, V$$

basado en que es el resultado de la “dificultad” y el “volumen” (del esfuerzo intelectual!). A partir de esta ecuación, propone que el “tiempo de desarrollo” del programa se puede calcular con una nueva hipótesis de tipo psicológico según las ideas de **Stroud**:

> **hay aproximadamente 10 momentos de tiempo psicológico por cada segundo de tiempo físico, pero pueden llegar a tantos como 20 o a tan pocos como 5**

El “tiempo de desarrollo” de un programa es la suma de la cantidad de “momentos de tiempo psicológico” empleados en el “esfuerzo” total. De aquí Halstead obtiene que el tiempo, en segundos, es:

$$T = \frac{E}{18}$$

El número 18, que divide al “esfuerzo” para obtener el “tiempo de desarrollo” de un proyecto, es el valor elegido entre 5 y 20 segundos, según los datos de Stroud. Finalmente, otra fórmula estima el número de “errores” del programa como:

$$\frac{V}{3000}$$

La teoría de Halstead, contemplada con los ojos de la presente década, se presenta como una colección de arbitrariedades sin fundamento. No obstante este hecho, todavía hoy continúa siendo citada y analizada. Ha sido el esfuerzo teórico más inspirador de todos cuantos se han realizado.

2.5 Complejidad ciclomática (McCabe)

Otra métrica abstracta que ha tenido importancia en la ingeniería del software es la complejidad ciclomática de McCabe. Para este autor, la complejidad de un programa no se basa en la manera como se usan operandos y operadores sino en la estructura lógica de los programas.

Esta métrica se basa solamente en el flujo de control de un programa. Si se abstrae todo el código del programa y solamente tenemos en cuenta los puntos de bifurcación de la secuencia de instrucciones, podemos dibujar un grafo que describe la estructura lógica interna. La métrica actúa sobre este grafo.

Sea ***e*** en número de caminos del grafo lógico del programa. Sea ***n*** el número de nodos del grafo. La complejidad ciclomática se define como:

$$c = e - n + 1$$

Este valor se lo puede interpretar como el número de decisiones, más una, que tiene el programa. Según McCabe un módulo bien estructurado tiene entre 3 y 7 de complejidad. Considera que 10 es el límite superior razonable que se puede esperar.

Esta métrica es una especie de “otra cara” de la métrica de Halstead. No obstante su carácter eminentemente teórico, ha sido normalizada por el IEEE.

2.6 Estudio de casos con métricas teóricas

Caso 1: Aplicación de la métrica de Halstead

Consideremos un programa clásico que calcula factorial de N. El lenguaje empleado es un seudocódigo cualquiera:

```
input N
if N=<0 then
      F=0
      return F
else
      F=1
      while N>1 loop
           F=F*N
           N=N–1
      end loop
      return F
end if
end
```

A los efectos de la métrica de Halstead hay que comenzar por contar los operadores y los operandos. La lista de operadores y operandos completa, sin mayor rigor crítico, es:

*	0
–	0
=	1
=	1
<	1
>	F
else	F
end	F
end	F
end	F
if	F
if	N
input	N
loop	N
loop	N
return	N
return	N
then	
while	

Puede observarse que los operadores y operandos prácticamente coinciden en número y prácticamente alternan entre ellos a lo largo del programa: estas hipótesis están dentro de esperado por Halstead. De aquí resultan los valores numéricos:

$n_1=13$	número de operadores diferentes de un programa
$n_2=4$	número de operandos diferentes de un programa
$N_1=19$	número total de operadores que aparecen en el programa
$N_2=17$	número total de operandos que aparecen en el programa

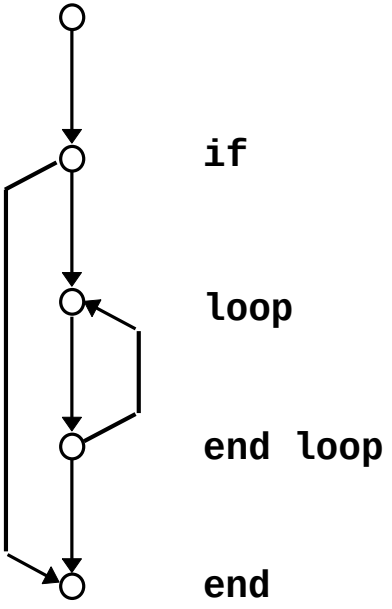
De estos valores se pueden estimar las diferentes métricas de Halstead:

“vocabulario” del programa	17	
“largo” del programa	36	
largo teórico del programa	56,11	bits
“volumen” del programa	147,15	
“dificultad” del programa	27,63	
“esfuerzo” del programa	4064,98	
“tiempo de desarrollo”	225,83	segundos
“errores” del programa	0,05	

Sin duda que el tiempo de desarrollo, algo menos de cuatro minutos, es uno de los resultados más graciosos de este cuadro de valores.

Caso 2: Ejemplo de métrica de McCabe

Consideremos el ejemplo del Caso 1. Apliquemos la métrica de McCabe. Para esto debemos realizar el diagrama de flujo de la información del programa. Para esto consideramos las instrucciones de bifurcacion y armemos el grafo de secuencia lógica:



Este grafo tiene 5 nodos y 6 caminos, su complejidad ciclomática es 2, lo cual evidencia la sencillez del módulo.

3. MÉTRICAS EMPÍRICAS DE PROYECTOS

3.1 Introducción

En la ingeniería del software se emplean casi siempre métricas empíricas. Las métricas teóricas, en general, solamente se aplican a **algoritmos** y no a sistemas de información. Puede decirse que poseen interés en campos específicos de aplicación, por ejemplo, en los sistema de Tiempo Real.

Las métricas empíricas se han basado, al menos al comienzo, en diferentes características **internas** o **externas** del proyecto:

- El tamaño final del proyecto.
- La cantidad de módulos.
- El tamaño del programa ejecutable final.
- La cantidad de líneas de código.
- La medida de las funcionalidades.

De todas estas propuestas, las únicas que realmente poseen interés, desde el punto de vista de la ingeniería, son las dos últimas: la cantidad de líneas de código como medida interna y las funcionalidades como medida externa. A ellas nos dedicaremos en este capítulo introductorio.

3.2 La cantidad de líneas de código

La cantidad de líneas de código era la métrica favorita de las décadas del 70 y 80. **Todavía hoy es una importante metodología en uso**. Las líneas de código son simples de contar y suministran una excelente información sobre los módulos, programas y sistemas.

Por cierto que la noción de líneas de código requiere algunas precisiones. Para comenzar, se trata de líneas de código de la programación **fuente**. Esto

ya pone una frontera a la aplicación de esta métrica. Con la aparición de las herramientas CASE, las líneas de código fuente comenzaron a perder importancia y a ser, cada vez más, el resultado de la actividad de un programa generador de programas. Esta es la razón por la cual esta métrica perdió aplicación en la década del 90. No obstante esto, el número de líneas **COBOL** de un proyecto sigue siendo un standard de hecho en la ingeniería del software.

Una segunda precisión es que se trata de contar líneas **finales** de la programación fuente. Todo aquel código que se empleó para ensayos, que fue desechado, que servía para conversión de datos por única vez, el código de prototipos y otra programación similar, no se debe contar como parte del sistema. Solamente la versión final es la que importa.

Una tercera precisión es que se deben contar líneas finales **efectivas** de la programación fuente. De hecho, las líneas vacías o los comentarios se deben dejar de lado. A veces no es simple eliminar o no contar estas líneas: en todo caso puede realizarse una estimación del porcentaje de líneas efectivas dentro del total.

Una cuarta precisión es que en muchos lenguajes es posible, según el estilo del programador, escribir más o menos líneas para exactamente el mismo resultado. El caso extremo es **C**, lenguaje en el cual se podría (pero es un horror desde el punto de vista de la ingeniería) escribir cualquier programa en una única línea.

Por lo expuesto, es claro que la métrica de las líneas de código tiene claras ventajas e inconvenientes.

A su favor cuenta que es sencilla de usar y que existe una gran cantidad de información empírica publicada.

En su contra cuenta que es fuertemente dependiente de la tecnología: el número de líneas de un sistema depende de la tecnología de implementación (además del **estilo** del programador). Es difícil de emplear en estimaciones preliminares: las líneas de código son un claro dato interno del proyecto; se las conoce con toda precisión al finalizar, pero no hay maneras sencillas de

realizar una estimación inicial en base a los datos externos del proyecto. Finalmente, carece de significado para las metodologías **CASE** y para los lenguajes visuales.

3.3 Los Function Points

Allan J. Albrecht de IBM, introdujo esta metodología en 1979. Desde ese momento esta métrica no ha dejado de ganar terreno.

Esta métrica es una de las pocas que está claramente orientada a los datos externos de un proyecto. Pretende medir **las funcionalidades que ve el usuario** del sistema de información. Por esta razón se debería traducir al castellano como **puntuación de las funcionalidades**, pero es habitual llamarla Puntos Funcionales o Puntos Función.

Fue diseñada para sistemas de información clásicos y, hasta el momento, este es su único campo de aplicación.

La métrica se basa en cinco funcionalidades básicas.

- Número de diferentes tipos de entradas externas.
- Número de diferentes tipos de salidas externas.
- Número de consultas.
- Número de archivos lógicos usados.
- Número de interfaces a otros sistemas.

A partir de estas cantidades se asignan puntos (elegidos por Albrecht) a cada una, según el siguiente esquema de cálculo:

Cuadro 1: Cálculo de Puntos Funcionales sin ajustar

Número de entradas	x 4 =
Número de salidas	x 5 =
Número de consultas	x 4 =
Número de archivos internos	x 10 =
Número de interfaces externas	x 7 =
Total (sin ajustar)	suma

En revisiones posteriores se consideró conveniente clasificar las funcionalidades en tres niveles: simples, promedio y complejas. De esta manera se extendió el cuadro del cálculo a una suma más compleja, del tipo:

$$PF_{sa} = \sum eE + \sum sS + \sum cC + \sum aA + \sum iI$$

En esta expresión se designa con ***E, S, C, A, I*** respectivamente a la cantidad de cada una de las cinco funcionalidades, en tanto que las minúsculas representan los pesos asignados a cada funcionalidad.

A los efectos de tener en cuenta otros aspectos del proyecto, Albrecht propuso, en su revisión de 1984, el empleo de 14 características modificadoras:

Comunicaciones.	Actualización on–line.
Performance.	Procesamiento complejo.
Funciones distribuidas.	Reusabilidad.
Configuración muy usada.	Facilidad de instalación.
Frecuencia de transacciones.	Facilidad de operación.
Entradas on–line.	Múltiples lugares.
Eficiencia para el usuario.	Facilidad de cambio.

A cada uno de los modificadores se le asigna una importancia en puntos comprendida entre 0 y 5. Cada punto asignado agrega un 1% a un factor de ajuste mediante la ecuación lineal:

$$f = 0.65 + 0.01\ S$$

donde S es la suma de todos los valores asignados a los 14 modificadores.

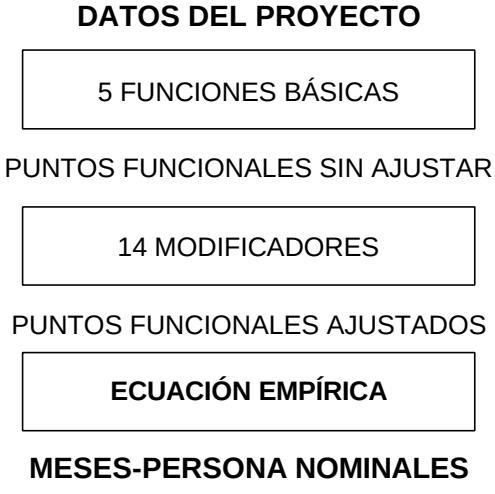
De este modo se llega a la puntuación corregida de las funcionalidades:

$$PF_a = f\,.\,PF_{sa}$$

En los planes originales de Albrecht, esta medida servía como dato de un estimador empírico. Hoy ya no se lo considera más así y la noción de Puntos Funcionales se ha separado de toda metodología de estimación de esfuerzo.

3.4 Metodología original de Albrecht

Albrecht buscaba un estimador empírico confiable. Su metodología seguía el siguiente esquema:



La ecuación empírica de Albrecht solamente valía para sistemas realizados en COBOL. No cabe duda que sus ecuaciones continúan siendo válidas hoy (excepto por algunas objeciones que analizaremos más adelante), pero el éxito de la métrica sobrepasó por mucho a su ecuación empírica.

Hoy, la metodología de los Puntos Funcionales tiene gran vigencia y se procura que no se mezcle con ecuaciones empíricas de estimación de esfuerzo.

3.5 Problemas del ISO 9000

Los sistemas de gestión de la calidad, tal como los exige la norma ISO 9000, necesitan metodologías bien definidas y bien documentadas. Estas

exigencias se extienden a las métricas para estimar, replantear o contratar proyectos.

Es por esta razón que el **IEEE** ha normalizado algunas métricas. Dentro de las métricas normalizadas se encuentran:

- Medida de la cantidad de líneas de código.
- La complejidad ciclomática.
- Los Puntos Funcionales.

Esta es una razón adicional para emplear estas métricas en el futuro.

3.6 Estudio de casos con métricas empíricas

Caso 3: Cálculo simple de Puntos Funcionales

A los efectos de introducir el tema, consideremos un caso simple de cálculo de Puntos Funcionales. Supongamos un sistema (de pequeñas proporciones) que posee las siguientes características:

10 archivos lógicos internos y ninguna comunicación con otro sistema.
30 programas para realizar la entrada, listado y consulta, es decir, el mantenimiento de los datos de cada uno de estos archivos.
15 programas de entrada de datos propios de la aplicación.
10 programas de listado específicos de la aplicación.
10 programas de consultas específicas propias de la aplicación.

En estas condiciones, la planilla de cálculo básico tiene el siguiente aspecto:

Número de entradas	25	x 4 =	100
Número de salidas	20	x 5 =	100
Número de consultas	20	x 4 =	80
Número de archivos internos	10	x 10 =	100
Número de interfaces externas	0	x 7 =	0
Total (sin ajustar)		suma	**!Syntax Error,)**

Este sistema de procesamiento de información, de proporciones modestas, tiene una cuenta de Puntos Funcionales sin ajustar (tomada según la versión original de Albrecht) de 380 puntos. En la versión estándar el cálculo requiere de mayor información acerca de estos elementos.

Caso 4: Replanteo de un proyecto grande (Puntos Funcionales viejos)

Este caso corresponde a un sistema real, ver el Caso 9, sobre el cual se realizaron las siguientes estimaciones:

Número de entradas	120	x 4 =	480
Número de salidas	200	x 5 =	1000
Número de consultas	30	x 4 =	120
Número de archivos internos	310	x 10 =	3100
Número de interfaces externas	60	x 7 =	420
Total (sin ajustar)		suma	**!Syntax Error,)**

En este caso tiene sentido aplicar esta estimación tan simple por falta de información más precisa. Es claro que un proyecto de este tamaño debe ser analizado con datos más precisos y con la metodología normalizada.

4. EL EMPLEO DE LÍNEAS DE CÓDIGO

4.1 La cantidad de líneas de código

La cantidad de línea de código es una métrica que todavía posee vigencia en alguna situaciones. Entre otras podemos considerar:

- Cuando se emplea un lenguaje de segunda a cuarta generación.
- Cuando se replantea un proyecto viejo realizado con estas técnicas.
- Por razones de comparación con otros datos.

En este capítulo se analiza con algo más de detalle los problemas de la medida de líneas. Comencemos por la definición de la unidad de medida. Es universal emplear como unidad de medida:

KLOC

que significa **miles de líneas** de programación objeto, finales, sin comentarios. Es usual aclarar, como información complementaria, el lenguaje en el cual se han escrito estas líneas.

4.2 Usos de las líneas de código

Las líneas de código en el caso de replanteo de proyectos es una técnica de aplicación **muy precisa**. Esta es la situación que ocurre en dos casos de mucho interés:

- En la emigración de un sistema existente (**heritage software**) a una nueva tecnología.
- En la extensión de un sistema existente, ya viejo, hecho con una tecnología casi abandonada.

Por el contrario, la métrica de las líneas de código es una tecnología de aplicación muy difícil en el caso de los proyectos nuevos puesto que las líneas de código que se terminará empleando no son fáciles de estimar al comienzo de un proyecto. No obstante esto veremos aplicaciones de la metodología.

Las línea de código dan medida útil para identificar propiedades globales de los proyectos. Algunos de los casos de interés son:

- Para la medida de complejidad para modelos de estimación empíricas (ejemplo típico es el modelo de **COCOMO**).
- Como unidad de medida clásica para indicar la **calidad** de una programación (cantidad de errores por KLOC).
- Medida de productividad, trabajo de mantenimiento o cualquier otro elemento asociado a la gestión.

También son útiles para la clasificación de tamaño de los proyectos.

4.3 El tamaño de los proyectos

La clasificación de proyectos en pequeños, medianos y grandes es una clasificación conceptual. Los proyectos **pequeños** consisten solamente en implementación y, por lo tanto, no tienen costos indirectos importantes. Por el contrario, los proyectos **grandes** además de la implementación poseen diversas fuentes de costos indirectos debido a la gerencia, control de calidad, capacitación de personal, mantenimiento, información para mercadeo y similares. Los proyectos **medianos** son intermedios entre los otros dos.

La frontera entre los diferentes tipos de proyectos no es precisa, pero la cantidad de líneas permite establecer una frontera cualitativa. Se supone que estamos en una situación donde la medida de cantidad de líneas es significativa del esfuerzo de programación y no en un caso de generación automática de programas. En estas condiciones, se pueden establecer las fronteras indicativas siguientes:

- Los proyectos pequeños poseen menos de **15 KLOC**.
- Los proyectos grandes poseen más de **100 KLOC.**

Sin duda esta limitación es dependiente de la tecnología, ya existirá oportunidad de definir otros criterios cuantitativos de interés.

4.4 Errores y líneas de programa

La medida de errores por líneas de programa es un resultado numérico de amplio uso y sobre el cual hay buena cantidad de datos.

El número de errores de la programación desciende a medida que se comprenden y se aplican mejor las técnicas de la ingeniería del software. Así por ejemplo, la NASA, que siempre se ha ocupado de controlar este parámetro, muestra que de una media de **9 errores por KLOC** a mediados de la década del 70 ha pasado a **6 errores por KLOC** en el presente.

En general, se acepta que una programación de buena calidad posee del orden de **1** error por **KLOC**. Menos errores significa que se han tomado cuidados extremos en la realización. Por el contrario, superar los niveles de 10 errores es entrar en la zona donde la calidad deja de ser razonable. Como criterio general, pueden aceptarse los siguientes niveles de defectos para proyectos medianos:

- una programación de **calidad razonable** posee menos de 10 defectos por KLOC.
- una programación de **buena calidad** posee cerca de 5 defectos por KLOC.
- una programación de **alta calidad** posee menos de 1 defecto por KLOC.

Vale la pena observar que las cifras presentadas son increíblemente altas, pero reales. Es un punto digno de ser tomado en cuenta.

4.5 Cantidad de líneas en proyectos nuevos

La cantidad de líneas es un dato interno de los proyectos. Por esta razón nunca ha sido simple estimar la cantidad de líneas que exigirá un proyecto nuevo.

La metodología básica consiste en emplear la experiencia previa bajo alguno de los siguientes aspectos:

- Extrapolar medidas de proyectos similares que se haya realizado en el pasado.
- Realizar un diseño preliminar que permita identificar los módulos existentes en el sistema.
- Emplear otras técnicas de medida de complejidad que se basen en datos externos y convertirlas en líneas de código, por ejemplo, emplear Puntos Funcionales como estimador auxiliar.

La información de proyectos previos permite tener cifras reales de las **KLOC** empleadas para resolver un determinado problema. Si ahora se compara el proyecto nuevo con algunos casos viejos disponibles, se pueden realizar las modificaciones en más o en menos que sean necesarias. Esta puede ser una buena estimación, pero depende de la experiencia de quien realice el análisis.

De estas metodologías, la más interesante de comentar es la segunda, la tercera es de aplicación inmediata. Un diseño de módulos de un sistema de información es algo que puede hacerse con un esfuerzo moderado. Si ahora disponemos de información acerca de la cantidad de líneas que tiene un módulo típico, es posible realizar una estimación preliminar.

El cuadro que sigue muestra cifras recogidas de la experiencia directa de un equipo de trabajo. Puede ser usado como valor indicativo.

Cuadro 2: Cantidad de líneas de código por módulo

	Líneas	Programas	Promedio
ASSEMBLER	14.400	299	48
RPG II	126.960	1.680	76
BASIC	2.953	33	89
COBOL	61.541	221	278

Es importante notar que en los datos de este cuadro está implícito un **estilo** de dividir en módulos y organizar un sistema de información. En rigor el cuadro que se debe usar es el que contiene datos del propio equipo de trabajo.

4.6 Ejemplo de modelo basado en líneas

A los efectos de considerar el tipo de modelos y ecuaciones empíricas que se analizaron en la década del 70 es interesante analizar un **ejemplo** concreto. Este ejemplo se ha elegido por una circunstancia particular, no porque sea un caso que pueda ser aplicado a la actividad práctica.

Los autores del modelo son Walston y Felix, de la empresa IBM. Es un modelo de tipo estático, con una variable exponencial. Las variables consideradas son:

Código	***KLOC*** miles de líneas de código final, fuente
Tiempo de desarrollo	***T*** meses
Esfuerzo	***K*** meses–persona

Las ecuaciones empíricas propuestas para los valores nominales son:

$$K = 5.2\ KLOC^{\,0.91}$$
$$D = 4.1\ KLOC^{\,0.36}$$

El modelo tiene en cuenta, además, un modificador de productividad que proviene de un estimador lineal empírico. Pero no interesa entrar en este tipo de detalles sino en el fondo de la cuestión.

Existen problemas con los modelos de tipo potencial o exponencial como el considerado. Una ecuación de esfuerzo del tipo:

$$K = c\ KLOC^{\,a}$$

(donde ***c*** y *a* son parámetros empíricos) plantea, de inmediato, una duda de tipo conceptual. Es usual que estos parámetros se ajusten mediante casos

reales. Luego de obtenidos los valores experimentales, según sea el valor del exponente *a* se tendrá:

- $a < 1$ efecto de economía de escala
- $a = 1$ un modelo lineal
- $a > 1$ efecto de **deseconomía** de escala

Existe una tendencia espontánea a suponer que la mayoría de los fenómenos de tipo económico poseen economía de escala debido a los ahorros logrados por la realización de algo más grande. También existe una tendencia natural a imaginar que los modelos lineales son exitosos. Sin embargo, la ingeniería del software nos reserva una enorme sorpresa.

4.7 ¿Economía o deseconomía de escala?

La mayoría de los autores que han estudiado el problema de la producción del software consideran que existe **deseconomía** de escala, excepto Walston & Felix y Nelson (que consideran exponentes muy próximos a 1 de todas maneras). Los exponentes que se proponen son tan bajos como **1.02** o tan altos como **1.83**, ver el Cuadro 3.

En el efecto de deseconomía de escala se encuentra la razón de fondo para diferenciar los proyectos pequeños de los grandes. También aquí se encuentra la causa por la cual las cifras de un proyecto pequeño conducen a errores enormes en un proyecto grande.

Cuadro 3: Exponentes empleados en ecuaciones de esfuerzo

MODELO	exponente
Walston–Felix, 1977	0.91
Nelson, 1978	0.98
Freburger–Basili, 1979	1.02
COCOMO (Organic)	1.05
Herd, 1977	1.06
COCOMO (Semidetached)	1.12
Frederic, 1974	1.18
COCOMO (Embedded)	1.20
Phister, 1979	1.275
Jones, 1977	1.40

Walston–Felix, 1977	1.43
Halstead, 1977	1.50
Schneider, 1978	1.83

La mayoría de los modelos del cuadro empleaban el número de línea de programa como variable principal para establecer la deseconomía de escala. Hoy se sabe que todas las demás métricas conducen al mismo tipo de resultados, esta no es una propiedad de las líneas de código sino del propio trabajo de desarrollo de proyectos de software.

Este resultado, claramente establecido en la década del 70, es la primera idea fundamental que se debe conocer acerca de la gestión de proyectos informáticos.

4.8 Causas de la deseconomía de escala

La deseconomía de escala de los proyectos de software proviene de varias fuentes simultáneas:

- La dificultad de comunicación entre los miembros del equipo de trabajo.
- La aparición de **nuevas funciones** en el proyecto y por lo tanto en el equipo de trabajo.
- La cantidad de errores **absolutos** que se tolera en un proyecto.

En todos los casos el esfuerzo aumenta más que proporcional al aumentar el tamaño del proyecto.

Las dificultades de **comunicación** entre miembros del equipo de proyecto es una de las causas más claras de la deseconomía de escala. A medida que aumenta el número de personas de un equipo, aumentan las comunicaciones de tipo técnico que un integrante debe comunicar a otros del equipo. Como consecuencia, aumenta el volumen dedicado a la documentación interna, aumentan las posibilidades de error y de discrepancia entre los módulos, todo lo cual aumenta el control de calidad.

Es interesante ver el problema desde el punto de vista teórico. Supongamos que ***N*** es el número de personas del equipo de proyecto. En un caso extremo, cada persona realiza un informe técnico acerca de su trabajo, que explica todos los problemas de interfaz con los módulos que tiene a su cargo. En el otro extremo de la complejidad, cada persona debe realizar una comunicación especial dedicada a cada miembro específico del equipo de trabajo que explica los detalles de las interfaces entre los módulos que tienen a su cargo. Esto indica que el esfuerzo de comunicación varía como ***N*** en el primer caso o como ***N.(N–1)*** en el segundo. De aquí que el esfuerzo de comunicación interna en un proyecto de software sea (posiblemente) un polinomio de segundo grado en ***N***. Este hecho explica porqué los exponentes del esfuerzo se encuentran entre 1 y 2, tal como muestra el Cuadro 3.

Las otras dos causas de la deseconomía de escala provienen también de la aparición de un esfuerzo adicional por la sola razón de aumento del tamaño. Así por ejemplo, la **documentación** que se exige, tanto técnica como de uso, aumenta en exigencias a medida que el proyecto es más grande. La **calidad de la programación** que se exige es mayor en un proyecto grande que en un proyecto pequeño. En un proyecto de 1000 líneas de código la cifra de 10 errores puede ser perfectamente aceptable. En un proyecto de 20 KLOC no es aceptable la misma proporción, porque daría 200 errores. Tal vez en este caso se deba llegar a un nivel de solamente 50 errores y esto supone 2,5 errores por KLOC. Esto hace que se deba multiplicar el esfuerzo de control de calidad en una forma no proporcional y hasta diseñar una estrategia para lograr esta meta tan exigente.

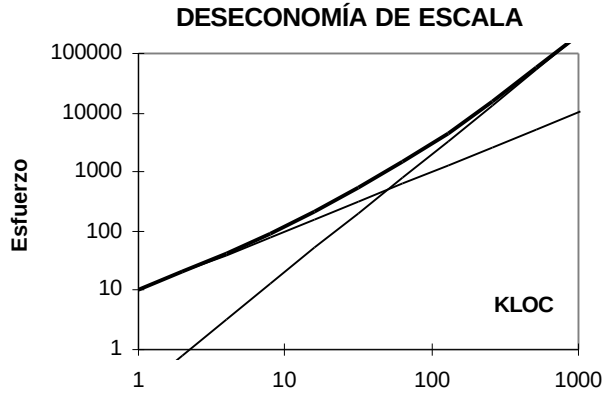
No es sencillo determinar la manera como actúa la deseconomía de escala. En los hechos, proyectos pequeños son aquellos donde el efecto no es sensible y proyectos grandes es donde la deseconomía de escala actúa en forma importante.

Regresando a la cantidad de personas (que es, a través del fenómeno de la documentación interna uno de los principales factores del fenómeno), podemos suponer que para KLOC pequeños el número de personas necesarios es lineal, pero al aumentar, por efecto de la comunicación mutua,

aparece un término de segundo grado. En esta interpretación, la expresión general del esfuerzo de proyecto sería:

$$K = a\ KLOC + b\ KLOC^2$$

y de allí la gran variabilidad de exponentes obtenido de coeficientes empíricos: están tratando de aproximar un polinomio de segundo grado con diversas “tangentes”.



Si la ecuación anterior tiene sentido, la zona de los proyectos pequeños es donde domina el primer sumando, la zona de los proyecto grandes es donde domina el segundo. Existe toda una zona intermedia, correspondiente a los proyectos medianos, donde ambos términos tienen influencia. En la figura (que es solamente un ejemplo), la zona comprendida entre 10 y 200 KLOC se está en la zona intermedia, antes domina el término lineal, después domina el término cuadrático.

4.9 Estudio de casos mediante líneas de código

Caso 5: Estimación de líneas de código en un proyecto nuevo

Se realiza un diseño preliminar de un sistema de información tal como el descrito en el Caso 3. La cantidad de módulo que se mencionan en este ejemplo es:

30 programas para realizar la entrada, listado y consulta, es decir, el mantenimiento de los datos de cada uno de estos archivos.
15 programas de entrada de datos propios de la aplicación
10 programas de listado específicos de la aplicación.
10 programas de consultas específicas propias de la aplicación

Esto supone un total de 65 programas. Aplicando las cifras del Cuadro 2 se tiene:

18.000 líneas de código COBOL
5.000 líneas de código RPG II

La diferencia de tecnología incide fuertemente en el número de líneas que tiene una aplicación, tal como muestra este ejemplo.

Para este caso se habían estimado 380 Puntos Funcionales sin ajustar, esto indica: 47 Puntos Funcionales por cada línea COBOL y 13 puntos por cada línea RPG II. Estos resultados son muy diferentes de los presentados en el Cuadro 13, entre otras razones, porque se trata de un análisis muy poco fino de los valores. Es interesante observar que para el cálculo de los Puntos Funcionales los archivos lógicos internos desempeñan un papel importante, en tanto que para el cálculo por módulos no son necesarios.

Caso 6: Líneas de código en la métrica de Halstead

Consideremos el Caso 1 como ejemplo de aplicación. En este ejemplo existe 1 operando por cada línea de programa en tanto que hay solamente 0.3 operadores por cada línea de programa. Al aumentar el tamaño del programa, el número de operandos no debería aumentar más allá de un valor propio de cada lenguaje de programación, en tanto que los operandos (variables, nombres de archivos, funciones, etc.) deberían crecer con la cantidad de líneas. En el ejemplo, la cantidad total de operadores es 1.3 veces el número de líneas de programa, pero esta cifra puede ser algo elevada.

De acuerdo a lo expuesto, podemos convertir las métricas de Halstead en aplicaciones de las líneas de código si realizamos algunas hipótesis simplificadoras:

- Aceptar la validez (hipotética) de la proporcionalidad entre los operandos y las líneas de programa (factor a).
- Aceptar la validez (hipotética) de la proporcionalidad entre los operadores y las líneas de programa (factor b).
- Aceptar que el total de operandos y operadores que aparecen es proporcional a la cantidad de líneas de programa (factores g y d).
- Aceptar que los tres factores de proporcionalidad son próximos a **1**.

En estas condiciones, para una programación de KLOC líneas de código, se tiene:

n_1 $1000\ a\ KLOC$
n_2 $1000\ b\ KLOC$
N_1 $1000\ g\ KLOC$
N_2 $1000\ d\ KLOC$

Para el “volumen” de un programa se tiene la expresión:

$$V = 1000\,(g + d)\,.\,KLOC\,.\log_2\,[1000(a + b)\,KLOC]$$

La “dificultad”, en las hipótesis que estamos haciendo, es proporcional al número de líneas, para la cual se tiene:

$$D = 500\,\frac{ag}{b}\,KLOC$$

De aquí resulta la ecuación de “esfuerzo” de Halstead:

$$K = 500000\,(g + d)\frac{ag}{b}\;\;KLOC^{\,2}\,\log_2\,[1000(a + b)\,KLOC]$$

Como puede apreciarse, Halstead obtiene (en las hipótesis hechas), un esfuerzo con deseconomía de escala de exponente 2 (además de un factor logarítmico): todo esto indica más deseconomía de escala que lo propuesto por los estudios empíricos. El "tiempo de desarrollo", que en los hechos es el esfuerzo de proyecto, en **horas–persona**, resulta:

$$T = 7.71\,(g + d)\,\frac{ag}{b}\;\; KLOC^{2}\,\log_2[1000(a + b)\,KLOC]$$

A los efectos de fijar las ideas, puede pensarse que a y b son del orden de 0.1 en tanto que g y d son del orden de 1. Con estos valores se tiene, como expresión estimada del esfuerzo:

$$T = 15.4\;\; KLOC^{2}\,[\log_2 KLOC + 7.64] \qquad \text{horas–persona}$$

Esta ecuación predice una deseconomía de escala muy importante. Por cierto que este caso es solamente a los efectos de mostrar las consecuencias teóricas de la métrica de Halstead, no posee importancia práctica. Es interesante aplicar la ecuación algunos tamaños de proyectos:

- Un proyecto pequeño, en el orden de 16 KLOC predice un tiempo de desarrollo de 179 KLOC ^{2} horas.
- Un proyecto mediano, en el orden de 64 KLOC predice un tiempo de desarrollo de 210 KLOC ^{2} horas.
- Un Proyecto grande, de 256 KLOC predice un tiempo de desarrollo de 240 KLOC ^{2} horas.

Todos estos tiempos no son comparables con la realidad práctica, son muchos mayores, por ejemplo, que los estimados por COCOMO. Finalmente, el número de "errores" de programa está dado por:

$$0.33\,(g + d)\,.\,KLOC\,.\log_2[1000(a + b)\,KLOC]$$

Aplicando los valores hipotéticos se tiene:

$$0.66\,KLOC\,.[\log_2 KLOC + 7.64]$$

Esta ecuación predice algo interesante: deseconomía de escala para los errores de programa, los errores aumentan en una forma más que proporcional con el número de líneas de código. Es interesante aplicar la ecuación algunos tamaños de proyectos:

- Un proyecto pequeño, en el orden de 16 KLOC predice 7,7 errores por KLOC.
- Un proyecto mediano, en el orden de 64 KLOC predice 9 errores por KLOC.
- Un Proyecto grande, de 256 KLOC predice 10 errores por KLOC.

En todos los casos, estas cifras son altas y corresponden a desarrollo de proyectos de relativamente baja calidad, pero no son cifras impensables.

5. EL MODELO COCOMO

5.1 Constructive Cost Model

El modelo COCOMO (COnstructive COst MOdel), modelo constructivo de los costos, es el modelo más exitoso dentro de las métricas que emplean el número de líneas de código como elemento fundamental.

Fue creado por Barry H. Boehm hacia 1980. Se basaba en el estudio de casos reales y su posterior análisis. Sus principales características son:

- es un modelo estático: estudia resultados globales
- emplea ecuaciones exponenciales de origen empírico
- emplea factores de costo como modificadores de ambiente

El análisis de casos lo lleva a Boehm a considerar tres tipos de proyectos desde el punto de vista de su análisis:

- Organic
- Semidetached
- Embedded

En el caso **Organic** se trata de un grupo pequeño de programadores expertos que desarrollan un proyecto en un ambiente que les es muy conocido. Esta es la situación habitual en los pequeños grupos de trabajo, en el trabajo de personas aisladas, estudiantes de cursos y situaciones similares. Usualmente el proyecto tiene un tamaño pequeño o mediano (decenas de miles de líneas de código).

En el caso **Embedded** se trata de un proyecto con condicionantes estrictas de todo tipo, que debe ser realizado con un costo determinado y entregado en una fecha precisa, para una plataforma determinada. Es un proyecto único

que posiblemente no se repita y sobre el cual no existe experiencia previa. Usualmente se trata de un proyecto mediano o grande.

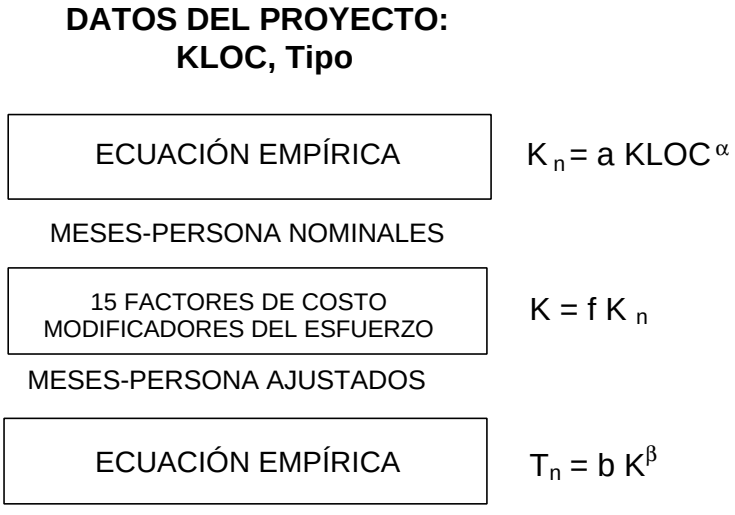
En el caso **Semidetached** se trata de una situación intermedia entre los dos extremos anteriores. Es claro que esta situación comprende la gran mayoría de los proyectos que se realizan en condiciones normales. En caso de dudas, éste debe ser el caso a aplicar.

5.2 Metodología de cálculo

La metodología COCOMO de cálculo comprende diversos pasos que se deben recorrer ordenadamente:

- Se toma como punto de partida las **KLOC** del proyecto. Una de las dificultades mayores consiste en realizar esta estimación.
- Se determina el tipo de proyecto considerado: Organic, Semidetached o embedded.
- Se calcula, mediante una ecuación empírica suministrada por el modelo, el esfuerzo **nominal K_n** (en meses–persona).
- Se estiman los 15 factores de costo considerados según las reglas del modelo. Se calcula el factor de ajuste **f**.
- Se calcula el esfuerzo **ajustado K** (en meses–persona).
- Se aplica otra ecuación empírica, suministrada por el modelo, para obtener el tiempo **nominal** o **sugerido T_n** de desarrollo del proyecto (meses).
- Mediante el esfuerzo y el tiempo de desarrollo se puede estimar el número de personas promedio que empleará el proyecto.

En el diagrama que sigue se ilustran los pasos a seguir, desde la información de entrada al modelo hasta los resultados obtenidos.



El modelo se basa en ecuaciones y coeficientes empíricos que permiten, a partir del número de líneas de código a desarrollar, estimar los parámetros del proyecto.

Es interesante observar que el **lenguaje de programación** no es dato explícito que se emplee, tal como se verán en lo que sigue. También es interesante observar que la metodología exige que la estimación del número de líneas del proyecto se realice mediante técnicas que no forman parte de la metodología COCOMO.

A partir del tipo de proyecto y del número de líneas de código se obtiene el esfuerzo nominal del proyecto en meses–persona. Esta cifra debe ser ajustada por los factores de costo.

A los efectos de la normalización de tiempos, en el caso de desear convertir los datos a horas, días o semanas, el modelo considera las siguientes relaciones:

- un mes–persona son **152 horas de trabajo** efectivas, a los efectos de tener en cuenta los días de licencia y de enfermedad
- un mes de trabajo tiene **19 días** efectivos

En lo que sigue se analizará en detalle el uso del modelo.

5.3 Los factores de costo

Los factores de costo son características del proyecto que modifican la estimación cruda del esfuerzo. Se determinan a partir de los siguientes conjuntos de atributos del proyecto:

- Atributos del producto
- Atributos de la plataforma
- Atributos del equipo de trabajo
- Atributos del proyecto

Cada atributo se debe evaluar en una escala entre 1 y 5. En algunos atributos es posible llegar al valor 6, en otros casos el valor 1, o aún el valor 2, carecen de interés. El valor 3 representa el caso promedio. Existen criterios para determinar el valor de cada atributo según se analiza en detalle en lo que sigue.

Para cada valor asignado, el atributo determina el valor de un factor de costo, que actúa como un coeficiente de ajuste. El ajuste completo se obtiene por el producto de todos los factores combinados. Como se puede experimentar fácilmente, colocando valores extremos en la planilla, los resultados pueden arrojar diferencias enormes en los factores de ajuste.

Cada factor de costo es identificado con un nombre simbólico que tiene importancia a los efectos del empleo de la planilla de cálculo. Estos nombres son los originales de Boehm y se indican entre paréntesis en la presentación que sigue.

5.4 Atributos del producto

Los atributos originados en el producto son tres:

- **Exigencias de confiabilidad (RELY)**: El valor 1 significa que los errores en el producto no tienen mayores consecuencias. El valor 3 significa que una falla provoca inconvenientes pero que son tolerables. El valor 5 significa que se exige confiabilidad máxima, porque existe posible riesgo de vida humana.

- **Tamaño de la base de datos (DATA)**: Este factor relaciona el tamaño de la base de datos con el tamaño del programa para manejarla. Se emplea una escala de tipo geométrico 10–100–1000 que determina los valores efectivos 2, 3 o 4, a aplicar. En forma cualitativa, el valor 1 significa una base de datos simple y sencilla de manejar; el valor 5 significa un diseño complejo con funciones de administración y recuperación importantes. La importancia de este punto tiende a ser disminuida por la aparición de nuevas tecnologías de manejadores de base de datos.

- **Complejidad del producto (CPLX)**: El valor 1 significa que es un producto muy simple, con algoritmos sencillos. El valor 6 significa que es un producto extremadamente complejo. La mayoría de la programación para uso comercial puede ser catalogada como normal y asignar el valor 3.

5.5 Atributos de la plataforma

Los atributos originados en la plataforma a utilizar son cuatro:

- **Exigencias de tiempo de ejecución (TIME)**: El valor 3 indica que el proyecto tiene exigencias normales, los valores menores carecen de importancia. El valor 6 indica un manejo complejo de recursos, prototipos, simulaciones, validación y todo lo necesario para cumplir con las exigencias de tiempo.

- **Exigencias de memoria (STOR)**: El valor 3 significa que la optimización del uso de la memoria no es importante. El valor 6 significa que se han tomado medidas extraordinarias para reducir el

empleo de memoria, se han realizado prototipos, simulación, validación y todo lo necesario.

- **Obsolescencia de la plataforma (VIRT)**: Este atributo caracteriza lo que sucederá durante el desarrollo y puesta en marcha del proyecto. El hecho que la plataforma (lógica o física) cambie a lo largo del proyecto agrega dificultades extraordinarias. El valor 2 indica que no esperan cambios. El valor 5 indica que es un elemento del proyecto el cambio de la plataforma y que este elemento deberá ser tenido en cuenta. Esta situación ocurre, por ejemplo, si el proyecto se extiende varios años en el tiempo.

- **Capacidad de respuesta del ambiente (TURN)**: Este parámetro mide la capacidad del ambiente de trabajo para simplificar el desarrollo. El valor 2 significa que se dispone de sistemas interactivos en los cuales es muy simple realizar cambios y nuevas versiones. El valor 5 significa que el sistema de desarrollo exige muchas horas para cada cambio, tal vez esperar hasta el otro día. Este atributo se vincula con la **plataforma de trabajo** y no con la plataforma objeto del desarrollo, es una observación que se debe tener en cuenta en los casos que no coincidan.

5.6 Atributos del equipo de trabajo

Los atributos del equipo de trabajo son cinco:

- **Capacidad de los analistas (ACAP)**: Este atributo mide la capacidad técnica y habilidad de trabajo **en equipo** de los analistas del proyecto. No se trata de una calificación individual sino de la capacidad de cooperación, comunicación y de realización como conjunto.

- **Experiencia en la aplicación (AEXP)**: Este atributo mide la experiencia del equipo en la aplicación que se considera. El valor 1 significa menos de 4 meses de experiencia. En valor 2 significa un año de experiencia. El valor 3 (valor promedio) significa **tres años** de experiencia en la aplicación. El valor 5 significa más de 12 años de experiencia.

- **Capacidad de los programadores (PCAP)**: Este atributo se aplica al equipo de programadores y no a las personas individuales. El valor 1 indica que existe un mala relación con los analistas y una pobre eficiencia de trabajo. El valor 5 indica la situación opuesta.

- **Experiencia en la máquina lógica (VEXP)**: Este atributo mide el conocimiento que se tiene de la máquina lógica sobre la cual se trabaja, no se trata del lenguaje de programación, el cual se tiene en cuenta en el atributo siguiente. El valor 1 significa menos de 1 mes de experiencia. El valor 2 significa unos 4 meses de experiencia. El valor 3 (valor promedio) significa un año de experiencia. El valor 4 significa más de 3 años.

- **Experiencia en el lenguaje de programación (LEXP)**: Este atributo mide el conocimiento del lenguaje de programación o de la herramienta que se emplea. El valor 1 significa menos de 1 mes de experiencia. El valor 2 significa unos 4 meses de experiencia. El valor 3 (valor promedio) significa un año de experiencia. El valor 4 significa más de 3 años.

5.7 Atributos del proyecto

Los atributos del proyecto son tres:

- **Empleo de técnicas modernas de programación (MODP)**: Se consideran técnicas modernas: análisis y diseño top–down, la documentación estructurada, el desarrollo top–down, el control de calidad durante el desarrollo, el código estructurado y el manejo sistemático de bibliotecas de módulos. El valor 1 significa que no se usan estas técnicas. El valor 3 significa que existe una cierta experiencia en su empleo. El valor 5 indica que todas estas técnicas son práctica de rutina.

- **Empleo de herramientas (TOOL)**: Este atributo mide el uso de herramientas que mejoran el rendimiento del desarrollo. El valor 1 significa herramientas elementales, que no pasan de un nivel simple.

El valor 3 significa que existen manejadores de bases de datos, depuradores de programa y herramientas interactivas. El valor 5 significa que existen herramientas especializadas para el proyecto.

- **Exigencias de plazo de entrega (SCED)**: Este atributo mide las exigencias de plazo de entrega comparadas con el tiempo nominal de proyecto que proponen las ecuaciones de COCOMO. De hecho, este modificador exige una cierta interacción en el cálculo. El valor 1 significa que existe un aumento de velocidad (realizar el proyecto en el 75% del tiempo nominal). El valor 2 supone un aumento moderado (realizar en 85%). El valor 4 supone que el proyecto se extenderá más de lo nominal (realizar el proyecto en un 130% del tiempo nominal). El valor 5 supone más del 160% del tiempo nominal. Es interesante observar que ambos extremos, acelerar o retardar el tiempo nominal, **penalizan** el esfuerzo del proyecto.

5.8 Uso de la planilla GPS

La planilla GPS posee todas las funciones necesarias para aplicar el modelo COCOMO. Estas funciones, con sus correspondientes ayudas, se las encuentra en el grupo de **Funciones definidas por el Usuario**. Por ser funciones de uso general, pueden ser empleadas en cualquier parte de esta planilla o de otra (en tanto esté abierta GPS). Las funciones se encuentran agrupadas en tres conjuntos:

- Las funciones de esfuerzo nominal.
- Las funciones de tiempo nominal.
- Los factores de costo.

Las funciones de esfuerzo nominal comienzan con la letra **K** y son tres. El argumento de estas funciones es **KLOC**, el número de miles líneas de código del proyecto.

Kemb	Esfuerzo nominal en un proyecto tipo Embedded.
Korg	Esfuerzo nominal en un proyecto de tipo Organic.
Ksd	Esfuerzo nominal en un proyecto de tipo Semidetached.

Las funciones de tiempo nominal comienzan con la letra **T** y también son tres. El argumento de estas funciones es **MesesPersona**.

Temb	Tiempo nominal de desarrollo en un proyecto tipo Embedded.
Torg	Tiempo nominal de desarrollo en un proyecto de tipo Organic.
Tsd	Tiempo nominal de desarrollo en un proyecto de tipo Semidetached.

Los factores de costo comienzan con la letra **x** Esta inicial se ha elegido a los efectos de encontrarlos juntos en la planilla, para evitar conflictos de nombres y para recordar que es un **factor**. El argumento de estas funciones es valor asignado al atributo correspondiente. Este valor usualmente está entre 1 y 5. **Si los argumentos no son valores enteros se devuelve cero**. En los casos que un valor está fuera de rango, de todas maneras la función devuelve el valor correspondiente al caso extremo. Así por ejemplo **xTIME** devuelve el valor 1.00 para los valores de los atributos menores o iguales a 3. La lista de factores de costo es:

xACAP	Capacidad de los analistas.
xAEXP	Experiencia en la aplicación.
xCPLX	Complejidad del producto.
xDATA	Tamaño de la base de datos.
xLEXP	Experiencia en el lenguaje de programación.
xMODP	Empleo de técnicas modernas de programación.
xPCAP	Capacidad de los programadores.
xRELY	Exigencias de confiabilidad.
xSCED	Exigencias de plazo de entrega.
xSTOR	Exigencias de memoria.
xTIME	Exigencias de tiempo de ejecución.
xTOOL	Empleo de herramientas.
xTURN	Capacidad de respuesta del ambiente.
xVEXP	Experiencia en la máquina lógica.
xVIRT	Obsolescencia de la plataforma.

A los efectos de simplificar el trabajo, existe una hoja que tiene el nombre COCOMO donde se han organizado los cálculos. En los casos de estudio se ilustra el manejo de esta hoja.

5.9 Estudio de casos mediante COCOMO

Caso 7: Proyecto pequeño analizado con COCOMO

Consideremos el ejemplo del Caso 5. Es claro que se trata de un proyecto pequeño de acuerdo con el número de líneas estimado:

18.000 líneas de código COBOL
5.000 líneas de código RPG II

Los parámetros del proyecto, calculados mediante la planilla GPS son, para la implementación en COBOL en las condiciones promedio:

Miles de líneas de programa	18		
	Caso considerado		
	Organic	Semi–detached	Embedded
Esfuerzo sin ajustar (meses–persona)	49,9	76,4	115,5
Factor de ajuste	1		
Esfuerzo ajustado (meses–persona)	49,9	76,4	115,5
Tiempo de desarrollo (meses)	11,0	11,4	11,4
Personas en promedio	4,5	6,7	10,1

En cambio, para la implementación en RPG II se tiene:

Miles de líneas de programa	5		
	Caso considerado		
	Organic	Semi–detached	Embedded
Esfuerzo sin ajustar (meses–persona)	13,0	18,2	24,8
Factor de ajuste	1		
Esfuerzo ajustado (meses–persona)	13,0	18,2	24,8
Tiempo de desarrollo (meses)	6,6	6,9	7,0
Personas en promedio	2,0	2,6	3,6

Sin duda en cualquiera de los dos casos nos encontramos en una situación Organic o Semidetached. En ambas situaciones es muy visible la diferencia de eficiencia de implementación a favor de RPG II, un lenguaje de mayor nivel que COBOL. Es claro que hay varios meses de diferencia en el tiempo de implementación, se emplea la mitad de la gente y el costo –por lo tanto– es bastante inferior. Estas son las razones de la popularidad del uso de RPG II frente a COBOL. Las estimaciones de COCOMO no hacen sino confirmar la publicidad acerca de esta herramienta.

También es interesante observar que si bien el proyecto es pequeño en su implementación RPG II, podría ser de tamaño mediano en su implementación COBOL debido el número de personas que emplearía en su desarrollo.

Caso 8: Proyecto mediano analizado con COCOMO

Se trata de una programación compleja, dedicada, destinada a un sistema de Tiempo Real realizada en C. El total del proyecto exigió 25 KLOC. El análisis de COCOMO, con sus correspondientes factores, realizado con la planilla GPS es:

Miles de líneas de programa	25	
	Caso considerado:	
	Organic	Embedded
Esfuerzo sin ajustar (meses–persona)	70,5	171,3
Factor de ajuste	0,878	
Esfuerzo ajustado (meses–persona)	61,9	150,5
Tiempo de desarrollo (meses)	12,0	12,4
Personas en promedio	5,2	12,1
FACTORES DE COSTO DEL PROYECTO	atributo	factor
Exigencias de confiabilidad	4	1,15
Tamaño de la base de datos	1	0,94
Complejidad del producto	5	1,3
Exigencias de tiempo de ejecución	4	1,11
Exigencias de memoria	3	1
Obsolescencia de la plataforma	1	0,87

Capacidad de respuesta del ambiente	0	0,87
Capacidad de los analistas	4	0,86
Experiencia en la aplicación	2	1,13
Capacidad de los programadores	4	0,86
Experiencia en la máquina lógica	4	0,9
Experiencia en el lenguaje de programación	5	0,95
Empleo de técnicas modernas de programación	4	0,91
Empleo de herramientas	2	1,1
Exigencias de plazo de entrega	4	1,04

Por las características del proyecto se lo puede considerar como a mitad de camino entre Organic y Embedded. Las dos estimaciones actuarán como casos extremos.

El plazo de entrega del proyecto es un año, cualquiera sea el caso. El número de personas es 5 o 12 según sea la situación. En los hechos, el proyecto se realizó con 7 personas, lo cual evidencia un buen ajuste con COCOMO. El esfuerzo total de proyecto se situó cerca de 100 meses–persona, también en excelente acuerdo con lo previsto.

Es importante observar la incidencia de los factores de ajuste sobre el resultado final. A pesar que existían varias condicionantes fuertes, el resultado final es un coeficiente de ajuste menor que 1.

Caso 9: Proyecto grande analizado con COCOMO

La programación de una empresa de seguros –realizada en COBOL– fue relevada a los efectos de estudiarla. El relevamiento arrojó la siguiente información:

Total de líneas de programa	732.538
Cantidad de programas	1.644
Archivos del sistema	310
Tablas asociadas	213
Menúes	125
Pantallas	122

Cantidad de registros	185
Claves	279

Analistas	8
Programadores	12
Asesores de gestión	2
Asesores técnicos	1
Personal de organización y métodos	2

Esta información es real, se la suministra completa a los efectos de estudiar el mismo problema mediante otras herramientas.

El primer punto interesante consiste en halla la media de líneas por programa, lo cual es una típica característica del **estilo** de trabajo. En este caso se tienen 445 líneas por programa que muestra una buena modularidad, si bien el promedio es algo alto.

El análisis de COCOMO, bajo condiciones medias indica:

Miles de líneas de programa	732,5	
	Caso considerado	
	Organic	Semidetached
Esfuerzo sin ajustar (meses–persona)	2444,9	4849,6
Factor de ajuste	1	
Esfuerzo ajustado (meses–persona)	2444,9	4849,6
Tiempo de desarrollo (meses)	48,5	48,7
Personas en promedio	50,4	99,5

Sin duda este sistema debiera ser considerado Semidetached, pero es claro que fue realizado en forma gradual, a lo largo de años, por lo cual la definición como Organic es más adecuada.

La cifra indicada para el equipo de trabajo es importante y contrasta bastante con el equipo efectivamente empleado. Contra las **50 personas** que sugiere COCOMO, el proyecto real ha empleado **25 personas** solamente. Esta es una situación típica de los países en vías de desarrollo.

Por el contrario, COCOMO sugiere **4 años** de tiempo de desarrollo –esta cifra y la cantidad de personas muestran que se trata de un proyecto grande– pero el proyecto real ha empleado cerca de **10 años**, no se tienen registros precisos sobre esta información. Por esta razón debe ajustaste el modificador de plazo de entrega y colocarlo en el nivel 5 (ninguna exigencia de plazo de entrega). Para este caso, la planilla muestra entonces:

Miles de líneas de programa	732,5
	Organic
Esfuerzo sin ajustar (meses–persona)	2444,9
Factor de ajuste	1,1
Esfuerzo ajustado (meses–persona)	2689,4
Tiempo de desarrollo (meses)	50,3
Personas en promedio	53,5

La situación que se analiza no corresponde verdaderamente a un único proyecto sino a una sucesión de proyectos y ampliaciones, muy dilatadas en el tiempo. Por esta razón COCOMO estima más personas y menos tiempo. Es posible, sin embargo, que de existir datos confiables, el total del esfuerzo estimado sea muy aproximado al esfuerzo real.

Si, por el contrario de lo que hemos hecho, analizamos este caso en sus tres módulos separados, tendremos de acuerdo con las cifras suministradas:

	líneas de código
Sistema 1	192.463
Sistema 2	294.910
Sistema 3	245.165

El análisis de COCOMO de los tres casos indica:

	Sistema 1	Sistema 2	Sistema 3
Esfuerzo ajustado (meses–persona)	660,9	1034,6	852,2
Tiempo de desarrollo (meses)	29,5	35,0	32,5
Personas en promedio	22,4	29,6	26,2

Este resultado muestra un excelente ajuste con lo que verdaderamente ha sucedido. Los tres sistemas se han desarrollado como tres proyectos sucesivos, ninguno de los cuales exige un número exagerado de personas y es bastante compatible con las 25 personas del equipo de trabajo. El tiempo total de desarrollo es de **97 meses** –o sea 8 años– cifra que también se encuentra muy próxima a lo que verdaderamente ha sucedido.

Finalmente, es muy importante observar que la suma de los esfuerzos de proyecto da 2548 meses persona, **algo menos** que si realiza el proyecto como un todo. Este es otro efecto de la deseconomía de escala.

Caso 10: Comparación de COCOMO con la métrica de Halstead

Es interesante comparar los resultados de la ecuación teórica de Halstead obtenida en las hipótesis del Caso 6:

$$T = 15.4 \quad KLOC^{\,2}\ [\log_{\,2} KLOC + 7.64] \qquad \text{horas–persona}$$

con los resultados (peores) que estima COCOMO. Como casos de estudio consideraremos los analizados en esta sección. Los resultados se presentan en el siguiente cuadro, en meses persona de 152 horas de trabajo:

	KLOC	Halstead	COCOMO
Caso 7	5	25,3	18,2
Caso 7	18	388,6	76,4
Caso 8	25	779,7	171,3
Caso 9	192	56997	660,9
Caso 9	245	94950	852,2
Caso 9	295	140027	1034,6

Resulta claro que esta métrica (con los valores de los coeficientes que se han tomado) predice valores **muy grandes** para el esfuerzo. Aún suponiendo que se eligen parámetros más pequeños que los del Caso 6 (*a* y *b* son 0.01 en tanto que *g* y *d* son 0.1) se tiene el cuadro:

	KLOC	Halstead	COCOMO
Caso 7	5	0,17	18,2
Caso 7	18	2,8	76,4
Caso 8	25	5,7	171,3

Caso 9	192	445,6	660,9
Caso 9	245	747,0	852,2
Caso 9	295	1106,7	1034,6

En este caso se tiene un buen ajuste en el caso de proyectos grandes, pero el efecto de deseconomía de escala se nota fuertemente en los proyectos medianos y pequeños.

6. LOS PUNTOS FUNCIONALES

6.1 Introducción

La metodología de los Puntos Funcionales o Puntos de Función es, sin duda, la métrica que tiene hoy mayores perspectivas tecnológicas. Existe alguna duda acerca de cómo se designan en castellano, en este libro se usará siempre Puntos Funcionales.

Como métrica, deriva de una propuesta, interna de IBM, realizada por Allan J. Albrecht en 1979. Esta metodología adquiere carácter público en 1981 y es revisada en 1984, fecha en la cual Albrecht introduce los modificadores. A partir de este momento gana aceptación creciente. Buena parte de su éxito se debe a que es una metodología **normalizada**. Debido a esta característica se puede esperar que personas diferentes, en países diferentes y con tecnologías de desarrollo diferentes obtengan resultado razonablemente coherentes.

La normalización es responsabilidad del International Function Point Users Group (IFPUG). Esta asociación de empresas usuarias es de libre afiliación y, además de la tarea de elaboración de la documentación normalizada, realiza reuniones periódicas, certifica técnicos y mantiene publicaciones regulares.

International Function Point Users Group
Blendonview Office Park
5008–28 Pine Creek Drive
Westerville, Ohio 43081–4899
Tel: (614) 895 7130
Fax: (614) 895 3466

El camino de la normalización fue largo y continúa como una actividad corriente:

- El **IFPUG** fue formado en 1986.
- La primera normalización del **IFPUG** es de 1990.
- La aceptación del **IEEE** es de 1992.
- El manual corriente es la versión 4.0, de Enero de 1994

Es muy importante seguir en forma estricta las recomendaciones de los documentos a los efectos de tener resultados comparables y compatibles.

6.2 Procedimiento de trabajo

El procedimiento de trabajo para contar Puntos Funcionales requiere seguir una serie de pasos bien definida:

- Determinar el tipo de cálculo a realizar.
- Identificar las fronteras del sistema.
- Contar las funciones tipo datos.
- Contar las funciones tipo transacciones.
- Calcular los Puntos Funcionales sin ajustar.
- Determinar los modificadores y el ajuste.
- Calcular los Puntos Funcionales ajustados.

En lo que sigue se analizarán punto por punto cada uno de estos pasos.

6.3 Tipos de cálculo de Puntos Funcionales

El primer paso de aplicación de la metodología consiste en determinar en cuál de tres situaciones nos encontramos:

- Un **proyecto nuevo**, a desarrollar o en desarrollo.
- **Ampliaciones** de un proyecto existente y ya completado.
- **Replanteo** de un proyecto existente y completado.

Las tres situaciones poseen interés. El caso del proyecto nuevo es la situación típica que se enfrenta todo jefe de proyecto. La ampliación de un

proyecto que se hizo aplicando la metodología es, sin duda, el caso más simple de todos: se trabaja en un terreno conocido y se tienen cifras previas.

El replanteo de proyectos ya completados tiene una importancia muy especial en los casos de emigración de un sistema viejo, con funcionalidades no despreciables y que ha desempeñado un buen trabajo durante años (**heritage software**), a una tecnología nueva. En este caso, el replanteo de lo hecho mediante la métrica de los Puntos Funcionales permite obtener una información muy valiosa a los efectos de la evaluación de la emigración, tal como se ve en los casos de estudio.

6.4 Las fronteras del sistema

La determinación de las fronteras del sistema en estudio es el primer paso del análisis. Tal como se analiza en otra sección, la cantidad de Puntos Funcionales de un sistema depende, de una manera no trivial, de la elección de las fronteras. Por esta razón se debe ser cuidadoso en este punto.

La frontera entre dos sistemas que interactúan se define desde **el punto de vista del usuario. N**o es una frontera técnica sino una frontera **funcional.** Cuando las fronteras no se conocen exactamente, deben ser estimadas, aunque para esto se deban emplear criterios convencionales.

Cuando se amplía un proyecto existente, las fronteras se deben considerar en forma coherente. De otra forma no es posible aplicar las ecuaciones y la metodología. En caso que la ampliación exija una redefinición de las fronteras, se debe replantear el proyecto existente de acuerdo con la nueva definición de las fronteras.

6.5 Las funciones tipo datos

Las funciones (o funcionalidades) de tipo datos comprenden todo lo que signifique almacenamiento dentro del sistema. Pueden pensarse como los viejos archivos, pero con una visión renovada. En el contexto de los Puntos Funcionales, archivo significa simplemente un **grupo de datos lógicamente vinculado**. No tiene nada que ver con la **implementación física** de las entidades.

Existen dos grandes grupos de archivos:

- Archivos lógicos internos (Internal Logical File, **ILF**): es un grupo identificable por el usuario de entidades de datos o de control que se encuentran vinculados lógicamente, **mantenidos** dentro de la aplicación.

- Archivos de interfaz externa (External Interface File, **EIF**): es un grupo identificable por el usuario de entidades de datos o de control que se encuentran vinculados lógicamente, **mantenidos** fuera de la aplicación

Para cada uno de los archivos es necesario determinar su complejidad. La complejidad depende de la organización lógica de los datos considerados. A estos efectos, las prácticas normalizadas consideran una manera objetiva, casi aritmética, de analizar la complejidad. Para esto se consideran dos tipos de conceptos:

- Tipo de elemento de datos (Data Element Type, **DET**): es un campo, no recursivo, que el usuario puede identificar como una unidad.

- Tipo de registro de datos (Register Element Type, **RET**): es un subgrupo de elementos de datos que el usuario puede reconocer como tal.

Consideremos, como ejemplo, la información que se almacena sobre un funcionario. Podemos reconocer en este almacenamiento, independientemente de la tecnología empleada, elementos tales como nombres, apellidos, direcciones, teléfonos, edades, y muchos más. Todos estos elementos son **DET**. A su vez, también dentro de esta información podemos reconocer estructuras tales como el funcionario con sus datos, la esposa del funcionario y los hijos del funcionario con los suyos. Todos estos elementos son **RET**.

A los efectos de normalización, existe un cuadro de complejidad que vincula estas dos características. Esta cuadro se aplica tanto a archivos internos como a interfaces externas al sistema.

La manera normalizada de proceder consiste en determinar el número de tipos de registros de datos (**RET**) y el número de tipos de elementos de datos (**DET**) empleados. Luego se usa este cuadro de complejidad:

Cuadro 4: Complejidad de los archivos lógicos

	de 1 a 19 DET	de 20 a 50 DET	más de 50 DET
sólo 1 RET	simple	simple	promedio
de 2 a 5 RET	simple	promedio	complejo
más de 5 RET	promedio	complejo	complejo

De esta manera pueden identificarse cada uno de los archivos lógicos internos o las interfaces externas y catalogarlas en sus tres categorías: simples, promedio o complejas.

6.6 Las funciones tipo transacciones

Las funciones (o funcionalidades) tipo transacciones comprenden las clásicas operaciones de entrada, salida y consulta de datos dentro de la aplicación. En el contexto de los Puntos Funcionales, estas operaciones significan simplemente operaciones sobre un **grupo de datos lógicamente vinculado**. No tiene nada que ver con la **implementación física** de las entidades.

- Entradas externas (External Input, **EI**): es un ingreso elemental de información de datos o de control, en forma unitaria, que proviene desde afuera de la aplicación.

- Salidas externas (External Output, **EO**): es un proceso elemental que genera información de datos o de control fuera la frontera de la aplicación.

- Consultas externas (External Inquiry, **EQ**): es un proceso elemental que combina una entrada y una salida de modo de recuperar información. No existe procesamiento de datos para la salida ni cambio de los archivos lógicos internos (excepto formato).

Para cada uno de las operaciones es necesario determinar su complejidad. La complejidad depende de la organización lógica de los datos considerados. A estos efectos, las prácticas normalizadas consideran una manera objetiva, casi aritmética, de analizar la complejidad. Para esto se consideran dos tipos de conceptos:

- Tipo de elemento de datos (Data Element Type, **DET**): es un campo, no recursivo, que el usuario puede identificar como una unidad.

- Tipo de archivo referido (File Types Referenced, **FTR**): es un archivo lógico o una interfaz que el usuario puede identificar como tal.

A los efectos de normalización, existe un cuadro de complejidad que vincula estas dos características.

La manera normalizada de proceder consiste en determinar el número de tipos de archivos referidos (**FTR**) y el número de tipos de elementos de datos (**DET**) empleados. Luego se usa este cuadro de complejidad de las entradas externas:

Cuadro 5: Complejidad de las entradas externas

	de 1 a 4 DET	de 5 a 15 DET	más de 15 DET
hasta 1 FTR	simple	simple	promedio
2 FTR	simple	promedio	complejo
más de 2 FTR	promedio	complejo	complejo

La manera de proceder con las salidas es similar. Consiste en determinar el número de tipos de archivos referidos (**FTR**) y el número de tipos de elementos de datos (**DET**) empleados. Luego se usa este cuadro de complejidad de las salidas externas:

Cuadro 6: Complejidad de las salidas externas

	de 1 a 5 DET	de 6 a 19 DET	más de 19 DET
hasta 1 FTR	simple	simple	promedio
de 2 a 3 FTR	simple	promedio	complejo
más de 3 FTR	promedio	complejo	complejo

La manera de determinar la complejidad de las consultas es consecuencia del hecho de tratarse de una operación doble. A estos efectos se procede como se indica:

- Se determina la complejidad de la parte de entrada de la consulta, mediante el cuadro correspondiente.
- Se determina la complejidad de la parte de salida de la consulta, mediante el cuadro correspondiente.
- Se emplea la mayor de las dos complejidades anteriores.

Procediendo de esta manera se puede clasificar cada una de las funcionalidades tipo transacciones en uno de los tres niveles. Esto completa el análisis de funcionalidades.

6.7 Vinculación con la métrica de Halstead

La métrica de los Puntos Funcionales no es solamente una feliz inspiración. Posee razones teóricas vinculadas con la vieja métrica de Halstead. Consideremos, como ejemplo, el caso de los archivos lógicos internos.

La métrica de Halstead se basaba en contar cantidades de operadores y operandos.

Los operadores que actúan sobre un archivo lógico interno se pueden asociar a la cantidad de **RET** que existen: cada una de sus estructuras determinará operaciones básicas. Los operandos se pueden asociar a la cantidad de **DET** que existen: la asociación es directa.

El cuadro de complejidad es una manera indirecta de calcular la métrica de Halstead aplicada a los elementos visibles por el usuario. Por esta razón, los Puntos Funcionales guardan una estrecha correlación estadística con la métrica de Halstead. Puede aceptarse que son una medida de la complejidad

del proyecto, medida en bits, según el punto de vista del usuario. Ya Albrecht había encontrado una alta correlación entre estos elementos.

6.8 Uso de la planilla GPS

La planilla GPS suministra todas las funciones necesarias para el cálculo de Puntos Funcionales sin ajustar. A estos efectos se han definido las siguientes funciones, cada una con tres argumentos (cantidad de elementos simples, promedio o complejos):

ILF Internal Logical File, Puntos Funcionales aportados por un archivo lógico interno.

EIF External Interface File, Puntos Funcionales aportados por una interfaz externa.

EI External Input, Puntos Funcionales aportados por una entrada externa.

EO External Output, Puntos Funcionales aportados por una salida externa.

No existe ninguna función para calcular las consultas puesto que se deben considerar como una entrada o una salida, según sea el caso más difícil. Esto no puede hacerse con una función sino que se debe analizar por separado cada porción de la transacción.

6.9 Estudio de casos de Puntos Funcionales sin ajustar

Caso 11: Calificar los archivos lógicos internos

Este caso es uno de los más representativos a los efectos de la clasificación en tres niveles. Consideremos un sistema dentro del cual se identifica el archivo de información de personas como una unidad lógica visible por el usuario.

Dentro del archivo podemos identificar, por ejemplo, dos grupos de información, lógicamente diferenciados para el usuario:

- información sobre la persona
- información sobre el lugar de trabajo

A los efectos de identificar la complejidad, se procede al análisis de DET y RET. La información sobre la persona contiene, por ejemplo:

- nombre de la persona
- fecha de nacimiento
- nombre del cónyuge
- nombre de los hijos
- cargo que ocupa
- fecha de ingreso al cargo
- calificación interna de la persona

En este caso tenemos 5 DET porque la información acerca del nombre de la persona tiene la misma estructura que la información del cónyuge o los hijos. Otra lista nos muestra que la información sobre el lugar de trabajo contiene:

- nombre de la empresa
- dirección
- ciudad
- localidad
- país

En definitiva, encontramos 6 DET. En resumen, para este archivo, existen 2 RET y un total de 11 DET, corresponde, por lo tanto, a un archivo **simple**.

Como puede apreciarse de este caso, es necesario una buena estructura o una gran cantidad de campos distintos para alcanzar el nivel promedio o complejo.

Caso 12: Replanteo de un proyecto grande

Consideremos el ejemplo del Caso 4, un proyecto grande que ha sido analizado con la definición vieja de los Puntos Funcionales. Los datos conocidos del proyecto son:

Número de archivos internos	310
Número de interfaces externas	60
Número de entradas	120
Número de salidas	200
Número de consultas	30

Para calcular los Puntos Funcionales sin ajustar es necesario dividir cada uno de las funcionalidad en sus tres niveles. Como esta información se desconoce podemos comenzar por colocarnos en las tres situaciones: todos simples, todos promedio y todos complejos. Por otra parte, a falta de información, podemos suponer que las consultas se dividen en partes iguales de cada tipo. Los valores obtenidos de la planilla GPS son, respectivamente: 3735, 5135, 7565 Puntos Funcionales.

Estas cifras defieren de 5120 que fue la obtenida en el Caso 4, pero **coincide** en a la hipótesis de suponer que todas las funcionalidades son promedio y todas las consultas del tipo entradas. Este resultado muestra que el IFPUG ha mantenido coherencia con la manera original de Albrecht de contar los puntos.

Una segunda observación es que hay una **considerable dispersión** de resultados si no podemos clasificar bien a las funcionalidades.

En un problema verdadero es necesario realizar la clasificación correspondiente, en este caso de replanteo se debe hacer alguna hipótesis adicional. En los sistemas de información típicos la distribución de funcionalidades responde a una distribución 80–20. La mayoría de las funcionalidades, un 80%, son simples el resto promedio o complejas. De acuerdo con esto, se puede armar el siguiente cuadro **estimativo**:

	simples	promedio	complejas
Número de archivos internos	248	50	12
Número de interfaces externas	48	10	2
Número de entradas	96	19	5
Número de salidas	160	32	8
Número de consultas	24	5	1

Consideremos los 310 archivos lógicos internos, un 80% deben ser simples y esto da 248. Luego quedan 62 para distribuir, de los cuales un 80% serán promedio y esto da 50, luego quedan 12 complejos. De la misma manera se procede en los restantes casos.

Con esta distribución de funcionalidades la planilla indica:

FUNCIONALIDADES DEL SISTEMA	simple	promedio	complejo	
ARCHIVOS LÓGICOS INTERNOS	248	50	12	2416
ARCHIVOS DE INTERFAZ EXTERNA	48	10	2	330
ENTRADAS EXTERNAS	96	19	5	394
SALIDAS EXTERNAS	160	32	8	856
CONSULTAS EXTERNAS (tipo entradas)	12	2	0	44
CONSULTAS EXTERNAS (tipo salidas)	12	3	1	70
Puntos Funcionales sin ajustar	**4110**			

Es interesante observar que el mayor peso del sistema se encuentra, en primer lugar, en los archivos lógicos internos y, en segundo lugar, en las salidas externas. Esta situación es típica de un sistema de información viejo como el considerado: almacena datos y genera informes, estas son sus principales funcionalidades.

Caso 13: Sistema de facturación simple

Se desear implementar un sistema simple de facturación contado. Se supone que el archivo de clientes se maneja en esta aplicación. Se supone que el archivo de artículos de se maneja en otra aplicación.

Para estimar el tamaño del sistema en Puntos Funcionales es necesario realizar un diseño preliminar. Para esto, uno de los mecanismos posible consiste en preparar una planilla electrónica como la que sigue:

Archivo de Clientes	ILF	S
Archivo de parámetros	ILF	S
Archivo de IVA	ILF	S
Archivo de Facturas	ILF	P
Comunicación con el archivo de Artículos	EIF	S
Ingreso de Clientes	EI	S

Ingreso de Parámetros	EI	S
Ingreso de IVA	EI	S
Ingreso de Facturas	EI	P
Impresión de Facturas	EO	C
Reimpresión de Facturas	EO	C
Resumen Diario	EO	S
Diario de Ventas	EO	P
Consulta de Clientes	EQ	S
Consulta de Artículos	EQ	S
Consulta de Facturas	EQ	P
Listado de Clientes por número	EO	P
Listado de Clientes alfabético	EO	P

En la primera columna se describe la funcionalidad que el sistema debe tener, en la segunda se la caracteriza como una de las cinco funcionalidades. En el tercera columna se establece si esta funcionalidad es simple (S), promedio (P) o compleja (C). En una primera aproximación, esta clasificación se puede hacer en forma intuitiva, pero en un análisis con mayor detalle debe salir de la consideración de DET, RET y FTR efectivos de cada caso.

El orden en que arma esta planilla es cualquiera, pero es usual comenzar por definir los archivos lógicos internos que el sistema debe tener. En este caso los archivos principales son: clientes, parámetros, IVA, facturas y artículos. Este último archivo, por las características del sistema de información, es una interfaz con otro sistema.

Luego de establecidos los archivos, se procede a definir los procesos de ingreso de datos. Cada archivo debe poseer uno o varios procesos de ingreso. Archivos y procesos de ingreso son el armazón del sistema de información. El objetivo es obtener información procesada.

El análisis preliminar se completa con se definen las salidas y las consultas del sistema que son la verdadera razón de existencia del sistema: almaceno información y proceso información para poderla recuperar o consultarla.

Una vez que esta planilla está completa, a juicio del diseñador, se puede ordenar por tipo de funcionalidades y por nivel de complejidad. De esta

manera se tiene la información de base para el cálculo de Puntos Funcionales sin ajustar. La planilla GPS (o la propia planilla, usando las funciones de GPS) nos da:

FUNCIONALIDADES DEL SISTEMA	simple	promedio	complejo	
ARCHIVOS LÓGICOS INTERNOS	3	1	0	31
ARCHIVOS DE INTERFAZ EXTERNA	1	0	0	5
ENTRADAS EXTERNAS	3	1	0	13
SALIDAS EXTERNAS	3	1	0	17
CONSULTAS EXTERNAS (tipo entradas)	0	0	0	0
CONSULTAS EXTERNAS (tipo salidas)	2	1	0	13
Puntos Funcionales sin ajustar	79			

Este caso de estudio nos muestra el tamaño de un sistema muy pequeño de manejo de información simple.

Caso 14: Sistema de facturación a crédito

Se considera el Caso 13 y se amplían sus prestaciones de modo de incluir:

- facturación a crédito
- devoluciones de compras
- venta en varias monedas
- comunicación el sistema de inventario
- comunicación con el sistema de deudores
- manejo de vendedores
- cierres mensuales de la venta
- generación de información estadística y de correo

Este es un caso de sistema de facturación más elaborado. El mecanismo de diseño preliminar y análisis de funcionalidades es el mismo. Por razones de sencillez, en la fase preliminar puede convenir que algunas funcionalidades no se detallen y solamente se indique cuantas se estiman que son. La planilla de archivos internos ahora contiene:

Archivo de Facturas Contado	C	1
Archivo de Facturas Crédito	C	1
Archivo de Devolución Contado	C	1
Archivo de Notas de Crédito	C	1
Archivo de Clientes	S	1
Archivo de parámetros	S	1
Archivo de IVA	S	1
Archivo de Vendedores	S	1
Archivo de Monedas	S	1
Archivo de Departamentos	S	1
Archivo de Términos de Pago	S	1

Esta planilla se complementa con los archivos de interfaz:

Comunicación con el archivo de Artículos	S	1
Comunicación con Sistema de Inventario	S	1
Comunicación con Sistema de Deudores	S	1

La planilla de funcionalidades de ingreso de datos es ahora:

Ingreso de Facturas Contado	P	1
Ingreso de Facturas Crédito	P	1
Ingreso de Notas de Crédito	P	1
Ingreso de Devoluciones de Contado	P	1
Ingreso de Clientes	S	1
Ingreso de parámetros	S	1
Ingreso de IVA	S	1
Ingresos varios	S	5

Observar que no ha interesado detallar diversos ingresos tales como datos de vendedores u otra información simple; solamente se estima que hay cinco en estas condiciones. La planilla de salidas de información del sistema es:

Impresión de Facturas Contado	S	1
Reimpresión de Facturas Contado	S	1
Resumen Diario	C	1
Diario de Ventas	C	1
Impresión y reimpresión de Notas de Crédito	C	2
Impresión y reimpresión de Facturas Crédito	C	2

Impresión y reimpresión Devolución Contado	C	2
Resúmenes de Fin de Mes	P	3
Listado de Clientes por varios criterios	P	4
Listados varios	S	5
Listado para Correo	S	1

En este caso se ha hecho uso de las posibilidades de no detallar demasiado los casos que son similares. Se puede observar que han aumentado bastante las salidas del sistema. Finalmente, la planilla de consultas es:

Depuración de Archivos	C	1
Consulta de Facturas	P	4
Consulta de Artículos	S	1
Consultas varias	S	5

Con estas planillas se puede realizar la clasificación final de funcionalidades para calcular los Puntos Funcionales sin ajustar:

FUNCIONALIDADES DEL SISTEMA	simple	promedio	complejo	
ARCHIVOS LÓGICOS INTERNOS	7	4	0	89
ARCHIVOS DE INTERFAZ EXTERNA	3	0	0	15
ENTRADAS EXTERNAS	8	4	0	40
SALIDAS EXTERNAS	8	7	8	123
CONSULTAS EXTERNAS (tipo entradas)	0	0	1	6
CONSULTAS EXTERNAS (tipo salidas)	6	4	0	44
Puntos Funcionales sin ajustar	317			

Caso 15: Comparación de diferentes sistemas de facturación

En el Caso 13 y el Caso 14 se analizaban sistemas de facturación con diferentes funcionalidades. El primer caso era muy simple, el segundo presentaba una elaboración adicional, pero de todas maneras sus funcionalidades están muy por debajo de las exigencias de un sistema comercial de cierta calidad.

Es interesante comparar tamaños de un mismo sistema, tal como es el casos de un sistema de facturación, bajo diferentes funcionalidades exigidas. Si

bien solamente hemos calculado Puntos Funcionales sin ajustar en los casos mencionados, la idea de esta comparación es examinar órdenes de magnitud y no cifras ajustadas.

Sistema de facturación simple	79
Sistema de facturación simple, a crédito	317
Sistema de facturación de pequeña empresa	1.000
Sistema de facturación corporativo pequeño	2.500
Sistema de facturación corporativo grande	7.500

No se puede realizar la pregunta ¿qué tamaño tiene un sistema de facturación? Porque no existe nada parecido a una respuesta. Según las funcionalidades del sistema, así será su tamaño. De todas maneras, un cuadro como el presentado ilustra los rangos de variación posibles. Con carácter general, los sistemas de información de pequeñas y medianas empresas se encuentran en las proximidades de 1.000 Puntos Funcionales en tanto que las versiones para grandes corporaciones tienen varios miles de puntos.

7. LAS CARACTERÍSTICAS MODIFICADORAS

7.1 Las características modificadoras

Las funcionalidades de un sistema de información –que conducen a los Puntos Funcionales sin ajustar– no tienen en cuenta dificultades o complejidades que el sistema puede tener debido a particularidades de diseño, implementación o consideraciones al margen de las funcionalidades. Estos aspectos del sistema de información se tienen en cuenta a través de las características modificadoras.

Las características modificadoras que permiten ajustar los Puntos Funcionales de un proyecto son catorce:

Comunicaciones.	Actualización on–line.
Performance.	Procesamiento complejo.
Funciones distribuidas.	Reusabilidad.
Configuración muy exigida.	Facilidad de instalación.
Frecuencia de transacciones.	Facilidad de operación.
Entradas on–line.	Múltiples lugares.
Eficiencia para el usuario.	Facilidad de cambio.

Cada una de estas características debe ser evaluada mediante una escala de 0 a 5. El valor típico o normal es 3, en algunos casos también 2 es un valor típico. A los efectos de la normalización, existen definiciones que permiten ajustar la cuenta de puntos a una práctica precisa.

En las secciones que siguen se analizará cada una de estas características en detalle.

7.2 Comunicaciones

Este modificador tiene en cuenta en qué medida el proyecto posee funcionalidades basadas en las comunicaciones. Cuanto mayores sean estas funcionalidades, más aumentará la cuenta final de Puntos Funcionales.

La definición normalizada de la escala de evaluación es la siguiente, según sean las características de comunicaciones del sistema en medición:

0 El sistema es batch o ejecuta en un PC autónomo.
1 El sistema es batch pero tiene entradas **o** salidas remotas.
2 El sistema es batch pero tiene entradas **y** salidas remotas.
3 El sistema tiene entradas on–line o teleprocesamiento.
4 Se ejecuta en varios procesadores pero existe un único protocolo de comunicaciones.
5 Se ejecuta en varios procesadores y existen varios protocolos de comunicaciones.

El valor 0 se aplica a un proceso sin comunicaciones. El valor 3 se aplica al caso normal: existe procesamiento on–line o teleprocesamiento. El valor 5 se aplica a una red compleja de máquinas, por ejemplo, donde coexisten máquinas UNIX, mainframes y redes locales. Los diversos protocolos agregan una complejidad general al sistema.

7.3 Performance

Este modificador tiene en cuenta las performance que se espera del sistema en consideración. El cuadro de evaluación es:

0 No existen exigencias de performance.
1 Se establecieron exigencias de performance pero no se tomó ninguna medida especial.
2 La performance es crítica en los momentos pico. No hubo diseño especial para cumplirlo. El procesamiento puede diferirse al día siguiente.
3 La respuesta es crítica en todas las horas de trabajo. No hubo diseño especial pero hay exigencias de comunicación con otros módulos.

4 Fue necesario un análisis de procesos en la etapa de diseño para cumplir con las exigencias.
5 Fue necesario emplear herramientas de análisis de performance para cumplir con las exigencias.

El valor 0 se aplica a un sistema sin exigencias de performance. Los valores 2 y 3 corresponde a situaciones típicas: hay exigencias críticas y el procesamiento puede diferirse o no al día siguiente. Los valores 4 y 5 se aplican a sistemas sumamente críticos, pocas veces encontrados en los sistemas de información habituales.

7.4 Funciones distribuidas

Este modificador tiene en cuenta en qué medida se trata de un sistema distribuido. Es un aspecto diferente de las comunicaciones: en aquel modificador importaba la complejidad de la comunicación, aquí importa el manejo de los recursos computacionales disponibles. El cuadro correspondiente es:

0 La aplicación no interviene en ningún procesamiento distribuido.
1 La aplicación prepara datos para que el usuario final los procese en forma independiente.
2 Los datos son preparados por la aplicación, transferidos y luego procesados por otra parte del sistema.
3 El procesamiento es distribuido pero la transferencia ocurre en una dirección solamente.
4 El procesamiento es distribuido y la transferencia es en ambas direcciones.
5 El procesamiento ocurre, dinámicamente, en el componente más apropiado del sistema.

El valor 0 es el caso en el cual no existen exigencias. La situación normal es 2 o 3: el sistema de información es distribuido pero existe solamente una transferencia simple de datos. El valor 5 se aplica solamente a un complejo sistema distribuido con asignación dinámica de recursos, situación muy rara. Esta situación no ocurre si la plataforma realiza esta operación sin que sea necesario diseñarlo específicamente.

7.5 Configuración muy exigida

Este modificador tiene en cuenta las exigencias de seguridad, de tiempos o de otro tipo que el sistema de información debe cumplir. El cuadro de valores es la siguiente:

0 No hay exigencias de ningún tipo.
1 Hay restricciones operacionales pero menos que lo normal. No es necesario nada para cumplirlas.
2 Se incluyen algunas condiciones de seguridad o de tiempos.
3 Se incluyen exigencias específicos para el procesador en algunas partes de la aplicación.
4 Existen restricciones específicas para el procesador principal o el procesador dedicado.
5 Existen, además, condiciones en las componentes distribuidas del sistema.

El valor 0 ocurre cuando no hay exigencias. El valor 5 se aplica al caso que existan condicionantes específicas, de seguridad o de tiempos, en cada una de las partes de la aplicación. La situación típica de un sistema de información es 2 o 3.

7.6 Frecuencia de transacciones

Este modificador tiene en cuenta la frecuencia de las transacciones y sus posibles picos. Es natural que cuanto más complejo sea el pico de carga a enfrentar, más complejo sea el sistema de información. El cuadro de valores correspondientes es:

0 No se espera ningún pico de carga.
1 Se espera un pico periódico (mensual, anual, etc.).
2 Se espera un pico semanal.
3 Se espera un pico diario.
4 Se requiere un análisis de tareas en las etapas de diseño del sistema.
5 Se requieren herramientas de análisis durante el diseño, el desarrollo o la instalación.

La situación normal es que exista un pico diario o semanal de carga, correspondiente a cierres de archivos o consolidación de datos. Los valores 4 y 5 se aplican a aquellos sistemas en los cuales es necesario un estudio e implementación especial para enfrentar el pico de carga.

7.7 Entradas on–line

Este modificador tiene en cuenta qué parte de las entradas son on–line. El cuadro es muy directo:

0 Todas las transacciones son batch.
1 Del 1% a 7% de las transacciones son interactivas.
2 Del 8% al 15% de las transacciones son interactivas.
3 Del 16% al 23% de las transacciones son interactivas.
4 Del 24% al 30% de las transacciones son interactivas.
5 Más del 30% de las transacciones son interactivas.

En los sistemas actuales de información es simple que se llegue a una puntuación alta en este modificador.

7.8 Eficiencia para el usuario

Este modificador tiene en cuenta las ayudas que encuentra el usuario en el sistema de información. Son ayudas elementos tales como: documentación en línea, barras de herramientas, menús pop–down, posibilidad de cambiar el idioma, atajos paras las operaciones frecuentes, etc. El cuadro de valores es de aplicación directa:

0 El diseño no incluye ningún tipo de ayudas.
1 El diseño incluye de 1 a 3 tipos de ayudas.
2 El diseño incluye de 4 a 5 tipos de ayudas.
3 El diseño incluye más de 5 tipos de ayudas.
4 Se requiere un estudio de eficiencia para el usuario.
5 Se requieren herramientas especiales para asegurar la eficiencia final.

En los sistemas modernos de información se tiende a disponer de diversas ayudas. De todos modos no es sencillo llegar al valor normal 3. Los valores 4 y 5 están reservados a los casos en los cuales se ha estudiado y diseñado cuidadosamente los aspectos de ingeniería humana del sistema. Puede tomarse como referencia de nivel 3 o superior las interfaces humanas gráficas de McIntosh, Windows o UNIX.

7.9 Actualización on–line

Este modificador presenta un aspecto diferente del que consideraba las entradas on–line. En este caso la preocupación es por la actualización on–line de los archivos y sus exigencias en casos de incidentes: errores humanos, cortes de energía u otros incidentes. El cuadro de valores es:

0 No hay exigencias.
1 Se actualizan entre 1 y 3 archivos. No hay exigencias especiales de recuperación.
2 Se actualizan más de 3 archivos. No hay exigencias especiales de recuperación en casos de incidentes.
3 Se actualizan todos los archivos importantes. Hay alguna exigencia de recuperación en caso de incidentes.
4 Es esencial la protección contra pérdida de datos y se la ha diseñado especialmente.
5 Debido a los volúmenes de información, el procedimiento de recuperación diseñado es muy cuidadoso.

El valor 0 se aplica a los casos donde no hay este problema. La situación normal, el valor 3, es que se actualicen los archivos importantes y que existan algunas exigencias de integridad de los datos. Los valores 4 y 5 se reservan para los casos donde la integridad de los datos es un punto importante y afecta el diseño y la implementación del sistema de información.

7.10 Procesamiento complejo

Este modificador tiene en cuenta la complejidad del procesamiento, elemento que no aparece en forma **directa** en el cálculo de Puntos

Funcionales. La calificación es igual a la cantidad de los elementos que siguen:

- Exigencias importantes de seguridad o controles de auditoría.
- Complejo procesamiento lógico.
- Complejo procesamiento matemático.
- Complejo manejo de excepciones: por transacciones incompletas, por datos perdidos, por errores humanos.
- Manejo complejo de la entrada/salida debido a necesitar independencia de dispositivos, multimedia, etc.

El complejo procesamiento lógico ocurre, por ejemplo, cuando hay un complejo manejo de bloqueo de registros que no se realiza automáticamente y debe ser diseñado. Las demás situaciones se describen por sí mismas.

7.11 Reusabilidad

Este modificador tiene en cuenta en qué medida el proyecto es más complejo debido a exigencias de reuso de la programación. En la medida que este es un objetivo, el proyecto es más complejo, aún para las mismas funcionalidades. Existen diversos niveles posibles que se definen en el siguiente cuadro:

0	El código no es reusable.
1	Se usa algo de código reusable dentro de la aplicación.
2	Menos del 10% de la aplicación tuvo en cuenta las necesidades de más de un usuario.
3	10% o más de la aplicación tuvo en cuenta las necesidades de más de un usuario.
4	La aplicación fue especialmente preparada y documentada para ser reusada. Se han adecuado los programas fuentes para ser adaptados a las necesidades de otros usuarios.
5	La aplicación fue especialmente preparada y documentada para ser reusada. Se han adecuado parámetros para ser adaptados a las necesidades de otros usuario.

La situación normal de reuso, niveles 2 y 3, corresponde a los casos en que existe una parte definida del proyecto que fue realizada pensando en su utilización posterior. Los niveles 4 y 5 corresponden a sistemas de información en los cuales se ha previsto su reutilización mediante cambios en el código o en parámetros.

7.12 Facilidad de instalación

La facilidad de instalación es una característica modificadora que aumenta la complejidad de un proyecto. La escala que se emplea es la siguiente:

0 No existen exigencias de instalación.
1 No existen exigencias del usuario pero hay condicionantes para la instalación.
2 El usuario fijó condiciones de instalación y conversión de datos, pero no inciden en forma importante en el proyecto.
3 Las exigencias de instalación y conversión de datos inciden en forma importante en el proyecto.
4 Se exige, además, herramientas de conversión e instalación automática.
5 Las herramientas de conversión fueron suministradas y probadas intensamente.

La situación normal corresponden a los niveles 2 y 3 y son claras. Los niveles 4 y 5 ocurren cuando existe programación automática de conversión e instalación. La diferencia entre ellos es el grado de prueba y de flexibilidad que estas herramientas poseen: es una experiencia cotidiana que aún los paquetes de programación utilitaria masiva, tales como editores de texto, tienen dificultades para enfrentar la instalación y la conversión automática cuando ocurren cambios de versiones.

7.13 Facilidad de operación

Esta característica modificadora tiene que ver con la administración del sistema y no con el usuario. La calificación es igual a la cantidad de los elementos que siguen que el sistema ha incorporado:

- Hay procesos de arranque, respaldo y recuperación que requieren la intervención del operador.
- Hay procesos de arranque, respaldo y recuperación que no requieren la intervención del operador.
- Se ha minimizado el cambio de medios de almacenamiento removibles.
- Se ha minimizado el empleo de papel.

Si la aplicación está diseñada para operación automática, sin operador, se califica como 5, situación que no es frecuente en los sistemas de información.

7.14 Múltiples lugares

Un sistema de información que se debe instalar en más de un lugar presenta dificultades adicionales para las mismas funcionalidades. La escala aplicable es la siguiente:

0 Existe un único lugar de instalación.
1 Se consideran múltiples lugares pero se opera bajo condiciones **idénticas**.
2 Se consideran múltiples lugares pero se opera bajo condiciones **similares**.
3 Se consideran múltiples lugares y se opera bajo condiciones **diferentes**.
4 Se ha elaborado y ensayado un plan para la operación en múltiples lugares **similares**.
5 Se ha elaborado y ensayado un plan para la operación en múltiples lugares **diferentes**.

Las situaciones normales corresponden a los valores 1 y 2. Lugares y condiciones diferentes quiere decir, por ejemplo, que el sistema está diseñado con una herramienta que puede trabajar en plataformas diferentes. Un ejemplo posible es un sistema que opera bajo PC, redes y máquinas UNIX. Las diferencias entre los valores 3, 4 y 5 contemplan diferentes grados de complejidad del diseño, pruebas y puesta en marcha de estas diferentes situaciones.

7.15 Facilidad de cambio

La facilidad de cambio es una característica para el **usuario**, no se relaciona con el reuso de la aplicación. La calificación es igual a la cantidad de los elementos que el sistema ha incorporado. Los elementos pertenecen a dos grupos:

- Consultas y reportes flexibles, simples, sobre un archivo (un punto).
- Consultas y reportes flexibles, simples, sobre más de un archivo (dos puntos).
- Consultas y reportes complejos sobre más de un archivo (tres puntos).

- ♦ La información de control está en tablas que pueden ser actualizadas y entran en operación al día siguiente (es necesario detener el sistema).
- ♦ La información está en tablas que pueden ser actualizadas y son operativas de inmediato (dos puntos).

El primer grupo de prestaciones no necesita aclaraciones adicionales. El segundo grupo tiene que ver con la capacidad del sistema de actualizar tablas de parámetros importantes. La posibilidad de modificar tablas que entran en operación de inmediato es bastante más que el ingreso on–line o la actualización de archivos que contemplan otros modificadores. Se trata aquí de parámetros que condicionan el diseño de archivos, con puntos de entrada de validez que están controlados. Exige aumento de complejidad del diseño lógico del sistema y por esto su característica modificadora.

7.16 Uso de la planilla GPS

La planilla GPS calcula el factor de ajuste a partir de los valores asignados a las características modificadoras. Cada punto asignado agrega un 1% al factor de ajuste mediante la ecuación lineal:

$$f = 0.65 + 0.01\ S$$

donde *S* es la suma de todos los valores asignados a los 14 modificadores.

7.17 Estudio de casos del factor de ajuste

Caso 16: Factor de ajuste de un sistema de facturación simple

Analicemos el factor de ajuste correspondiente al Caso 13 o al Caso 14. Se consideraban sistemas simples de facturación. Un análisis de los modificadores puede conducir a un cuadro de este tipo:

COMUNICACIONES	3
PERFORMANCE	2
FUNCIONES DISTRIBUIDAS	2
EXIGENCIA DE LA CONFIGURACIÓN	2
FRECUENCIA DE TRANSACCIONES	3
ENTRADAS ON–LINE	5
EFICIENCIA PARA EL USUARIO	1
ACTUALIZACIÓN ON–LINE	1
PROCESAMIENTO COMPLEJO	0
REUSABILIDAD	0
FACILIDAD DE INSTALACIÓN	0
FACILIDAD DE OPERACIÓN	1
MÚLTIPLES LUGARES	0
FACILIDAD DE CAMBIO	2

Con estas cifras, el sistema considerado en el Caso 14 se ajusta de la siguiente forma:

Puntos Funcionales sin ajustar	317
Coeficiente de ajuste	0,87
Puntos Funcionales ajustados	276

El hecho que el sistema sea muy sencillo y realizado en condiciones poco exigidas también se refleja en los modificadores: el resultado ajustado todavía es menor.

Caso 17: Modificadores de un heritage software

Consideremos el ejemplo tratado en el Caso 9. Se trataba de un proyecto grande, realizado en COBOL y comenzado hacia 1980. Se trata de reconstruir los modificadores y el factor de ajuste bajo el cual fue realizado este proyecto.

Es interesante preparas un cuadro con las condiciones que eran habituales para los sistemas de información hacia 1980 y con la condiciones que se exigen 15 años después al mismo sistema:

	1980	1995
COMUNICACIONES	0	3
PERFORMANCE	0	1
FUNCIONES DISTRIBUIDAS	1	2
EXIGENCIA DE LA CONFIGURACIÓN	0	0
FRECUENCIA DE TRANSACCIONES	0	3
ENTRADAS ON–LINE	0	4
EFICIENCIA PARA EL USUARIO	0	3
ACTUALIZACIÓN ON–LINE	0	3
PROCESAMIENTO COMPLEJO	2	2
REUSABILIDAD	0	0
FACILIDAD DE INSTALACIÓN	0	1
FACILIDAD DE OPERACIÓN	2	2
MÚLTIPLES LUGARES	0	0
FACILIDAD DE CAMBIO	0	3

Como es natural este cuadro refleja una opinión del autor para un caso concreto. No obstante esto es ilustrativa la comparación. Es claro que ha cambiado la manera de trabajar: de procesamiento casi exclusivamente batch se ha pasado a modalidades interactivas y esto se refleja en varios modificadores, incluyendo la eficiencia para el usuario.

Hay elementos que no se modifican y que tienen que ver con la complejidad y características propias del sistema: procesamiento complejo, reusabilidad, múltiples lugares, por ejemplo.

Hay condicionantes que aparecen como nuevas, tales como la facilidad de cambio, debido a un mundo con mayor movilidad.

En definitiva, comparando los coeficientes de ajuste para un sistema comercial típico se tiene:

	1980	1995
Coeficiente de ajuste	0,70	0,92

Esto quiere decir que el mismo sistema, con las mismas funcionalidades, tiene en 1995 un **30%** más de Puntos Funcionales que su implementación de 1980. Básicamente quiere decir que las mismas funcionalidades son hoy **más amistosas** que en 1980.

Caso 18: Comparación entre una empresa mediana y una corporación.

Supongamos que un sistema de información para una empresa mediana tenga las mismas funcionalidades que para una corporación internacional, cosa que es solamente un caso de estudio, no una situación real. Analicemos las diferencias de modificadores que estos presuntos sistemas iguales poseen.

	mediana	corporación
COMUNICACIONES	3	5
PERFORMANCE	3	4
FUNCIONES DISTRIBUIDAS	3	4
EXIGENCIA DE LA CONFIGURACIÓN	2	4
FRECUENCIA DE TRANSACCIONES	3	4
ENTRADAS ON–LINE	4	5
EFICIENCIA PARA EL USUARIO	2	5
ACTUALIZACIÓN ON–LINE	2	5
PROCESAMIENTO COMPLEJO	2	3
REUSABILIDAD	0	4
FACILIDAD DE INSTALACIÓN	2	5
FACILIDAD DE OPERACIÓN	2	4
MÚLTIPLES LUGARES	0	5
FACILIDAD DE CAMBIO	3	5

Las cifras de este cuadro reflejan la opinión del autor. En general obedecen a que el manejo de una gran corporación debe ser el mismo en múltiples situaciones diferentes, debe poseer una flexibilidad (por ejemplo por atender más de un idioma o más de una legislación) que el de una empresa mediana no posee. En un gran corporación tiene sentido invertir esfuerzo en optimizar la performance para el usuario y el administrador. En resumen, en todos los casos hay una exigencia algo mayor que contribuye a la cuenta final. De acuerdo con estas cifras, se tiene, para los coeficientes de ajuste:

	mediana	corporación
Coeficiente de ajuste	0,96	1.27

Aún aceptando que los sistemas puedan tener las mismas funcionalidades, las exigencias adicionales que tiene una corporación debido a su tamaño, distribución geográfica, manejo, auditoría y otros elementos, se traduce en un **30%** más de Puntos Funcionales. Sin duda es un punto de interés observar esta diferencia.

8. SUPERPOSICIÓN DE SISTEMAS

8.1 Las fronteras del sistema

La elección de las fronteras de un sistema no es un tema trivial o el resultado de una convención solamente. Las fronteras de un sistema y la separación con otros sistemas de información con los cuales se intercambia información es un punto delicado, tal como se verá en lo que sigue.

Si el problema se lo piensa en una forma poco crítica –tal como ocurre con la deseconomía de escala– se puede llegar a la conclusión que las fronteras no deberían ser relevantes en el análisis de un sistema de información. Cuando se piensa así se está suponiendo, implícitamente, que se puede aplicar lo que se llama un "principio de superposición" que es, en cierta medida, la misma idea de la linealidad entre el tamaño de un proyecto y el esfuerzo que exige realizarlo.

El sistema formado por la reunión o superposición de dos sistemas es algo diferente que cada una de sus partes. Esta "no linealidad" parece contradecir a la intuición, por esta razón realizaremos un estudio de casos que permita comprender el fondo del problema. Los casos a considerar tienen interacción creciente. A medida que la interacción entre los sistemas aumenta, se percibe con más claridad los problemas de la superposición y de la elección de las fronteras.

El problema se analizará para el caso concreto de la métrica de los Puntos Funcionales, donde la comprensión del problema es grande. No obstante esto, las ideas son generales y se reflejan en todas las métricas.

Como hipótesis simplificadora, se suponen que los sistemas considerados poseen los mismos modificadores. Esta hipótesis es correcta en el caso que los sistemas de información hayan sido diseñados con metodologías y estilos coherentes y formen parte de una misma instalación. Tal vez no sea cierta en

el caso que se hayan ensamblado sistemas parciales de diferentes orígenes (o de diferentes épocas) y que tienen, por lo tanto, criterios de diseño e implementación diferente.

En la hipótesis de coherencia de estructura, el factor de ajuste de los Puntos Funcionales puede manejarse como un factor común a todo el cálculo (los sistemas tienen una complejidad similar). De esta manera se simplifica mucho el análisis de los casos.

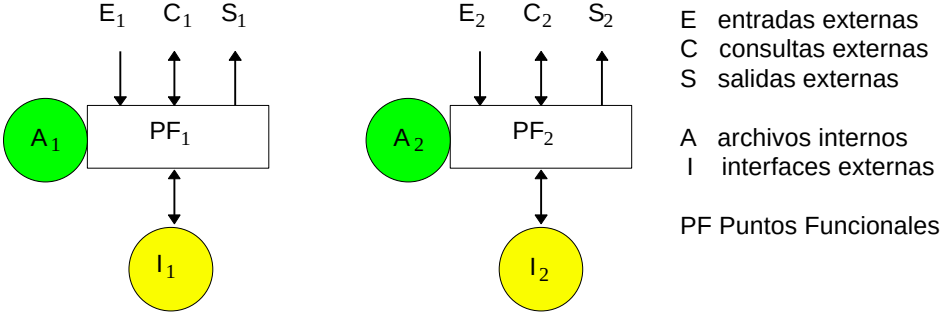
Los casos de interacción creciente que se considerarán son:

- Sin interacción: dos sistemas independientes.
- Interacción jerárquica: el modo cliente–servidor.
- Interacción fuerte: dos sistemas que interactúan entre sí.

De esta colección de casos podrán extraerse conclusiones generales de mucho interés teórico y práctico.

8.2 Dos sistemas independientes

Consideremos dos sistemas completamente independientes entre sí, esto es, que no intercambian información alguna. En el esquema que sigue se representan estos sistemas.



Podemos calcular ahora los Puntos Funcionales de cada uno de los sistemas y luego calcular como si ambos formaran un único sistema encerrado dentro

de una frontera ficticia. De acuerdo con la expresión general se tiene para cada uno de los sistemas:

$$PF_1 = \sum_1 eE + \sum_1 sS + \sum_1 cC + \sum_1 aA + \sum_1 iI$$
$$PF_2 = \sum_2 eE + \sum_2 sS + \sum_2 cC + \sum_2 aA + \sum_2 iI$$

Se ha indicado con **1** o **2** en las sumatorias, los elementos que deben ser sumados. Si consideramos ahora el sistema global, tendremos como expresión para los Puntos Funcionales del conjunto:

$$PF = \sum_{1,2} eE + \sum_{1,2} sS + \sum_{1,2} cC + \sum_{1,2} aA + \sum_{1,2} iI$$

Se ha indicado ahora con **1, 2** en las sumatorias, que se deben sumar los elementos de ambos sistemas. Se tiene, finalmente, separando las sumas en sus elementos correspondientes:

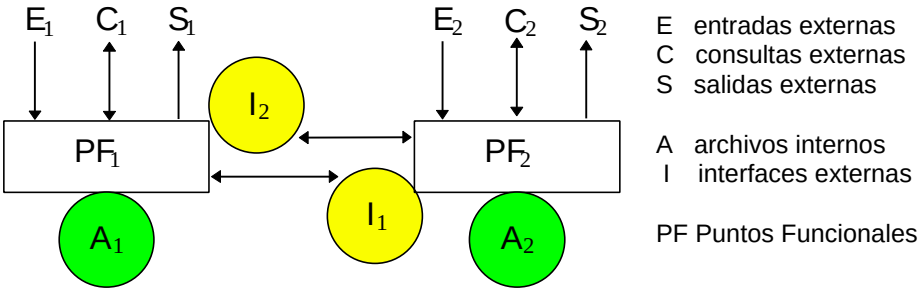
$$PF = PF_1 + PF_2$$

Esta expresión muestra que en el caso de sistemas autónomos, la cuenta de Puntos Funcionales obedece al principio de superposición: los puntos de una colección de sistemas son la suma de los puntos de cada sistema considerado individualmente.

Este resultado, si bien parece modesto, tiene mucha importancia para algunas aplicaciones prácticas que se verán más adelante.

8.3 Dos sistemas que interactúan entre sí

Consideremos dos sistemas que interactúan fuertemente, cada uno posee archivos lógicos que el otro usa, tal como indica la figura:



Las ecuaciones para cada uno de ellos son:

$$PF_1 = \sum_{1} eE + \sum_{1} sS + \sum_{1} cC + \sum_{1} aA + \sum_{2} iI$$

$$PF_2 = \sum_{2} eE + \sum_{2} sS + \sum_{2} cC + \sum_{2} aA + \sum_{1} iI$$

Si consideramos ahora el sistema formado por ambos, todos los archivos lógicos son internos y se tiene la ecuación:

$$PF = \sum_{1,2} eE + \sum_{1,2} sS + \sum_{1,2} cC + \sum_{1,2} aA + \sum_{1,2} aI$$

Se puede concluir entonces que:

$$PF = PF_1 + PF_2 + \sum_{1,2} (a - i)I$$

Esta ecuación muestra un resultado muy importante: los Puntos Funcionales del sistema formado por la reunión de los dos sistemas posee una cuenta mayor que cada uno de ellos por separado. En otras palabras, **el todo es más complejo o mayor que la suma de las partes**. Este resultado es el único principio de superposición válido y, en cierta medida, es la contraparte de la deseconomía de escala.

8.4 Conclusiones

Si bien toda la técnica de los Puntos Funcionales es lineal, no se cumplen las ecuaciones de superposición. El principio de superposición es válido solamente en el caso de sistemas que no interactúan. Es más, excepto en casos especialmente elegidos, en la mayoría de las oportunidades en que existe interacción, la cuenta de puntos de una colección de sistemas (o de subsistemas) es mayor que la suma de sus partes.

Una segunda conclusión importante es que la cuenta de Puntos Funcionales, por lo tanto, **depende de la elección de fronteras**. Este hecho era una de las primeras observaciones al introducir el tema. Este resultado no es casual. Fue una de las modificaciones que introdujo el **IFPUG** a la manera de contar los Puntos Funcionales original de Albrecht.

Esto puede ser resumido diciendo que, en la medida que un sistema está formado por partes, el total es más complejo que las partes involucradas. De otra manera: un sistema bien estructurado, con una estructura rica, posee más complejidad y, por lo tanto, mayor cantidad de Puntos Funcionales que un sistema hecho como un todo, sin estructura. La buena estructuración es una condición deseable pero esto no significa que no aumente la complejidad o tamaño del sistema.

8.5 Estudio de casos de superposición de sistemas

Caso 19: Puntos Funcionales de tres sistemas que interactúan

Consideremos un ejemplo (ficticio) de tres sistemas importantes que interactúan. El caso tiene por finalidad mostrar con números los efectos de principio de superposición.

Supongamos que el análisis de funcionalidades da, para los tres sistemas el mismo cuadro (todas las funcionalidades se consideran promedio):

Número de archivos internos	100

Número de interfaces externas	20
Número de entradas	40
Número de salidas	65
Número de consultas	20

Todo esto resulta que cada uno de los tres sistemas ofrece 10 archivos a los otros dos como interfaces externas, especialmente preparados para la comunicación con los otros sistemas. Este es un punto esencial. Si no se piensan las cosas así, por ejemplo porque se supone que cada sistema se las ingenia para extraer de los archivos del otro la información que necesita, entonces cada sistema tendrá él un archivo interno de comunicación, lo cual, en los lechos, es lo mismo.

Con estas cifras, cada uno de los sistemas posee, tal como calcula la planilla GPS, en la hipótesis que las consultas se dividen por mitades entre cada tipo, **1645** Puntos Funcionales sin ajustar.

Considerando ahora una única frontera del sistema, el cuadro es ahora:

Número de archivos internos	330
Número de interfaces externas	0
Número de entradas	120
Número de salidas	195
Número de consultas	60

Ya no existen interfaces, todos los archivos son internos. El hecho que hayan aparecido los 30 archivos que eran interfaces como archivos internos es el punto principal. El cálculo da ahora **5025** Puntos Funcionales que son más que **1645x3= 4935** de la cuenta de los sistemas por separado. Se pone claramente de manifiesto el fenómeno de que el todo es más complejo que la suma de sus partes.

9. PROYECTOS PARCIALES

9.1 Los proyectos parciales

En muchas oportunidades se plantea la necesidad de analizar un proyecto en forma parcial. Hay muchas situaciones diferentes, pero para mencionar solamente algunas situaciones tenemos:

- **Rediseño de un sistema**: el proyecto anterior actúa como prototipo del proyecto nuevo. El esfuerzo del total debería ser menor que si se lo hiciera completo.
- **Cambio de tecnología**: también en este caso el proyecto anterior actúa como prototipo (imperfecto) del proyecto nuevo. Su medida es una buena indicación de referencia.
- **Equipo de contraparte**: cuando existe un equipo de contraparte, este equipo debe ser considerado como parte del equipo total del proyecto. Es claro que parte del trabajo de gestión, control de calidad y aún partes del análisis son realizados por este equipo y, por lo tanto, forman parte del total del proyecto.
- **Ausencia de algún elemento**: puede ocurrir que el proyecto no posea algún elemento (instalación, documentación, etc.) En estos casos, según sea el tamaño del proyecto, se debe estimar la porción que no se realiza para quitarla del total de proyecto.

Los casos de proyectos parciales posee un tratamiento especial dentro de la metodología de los Puntos Funcionales. En el estudio de casos se presentan algunas situaciones.

9.2 Extensiones o cambio de tecnología de un sistema

En estos casos existen criterios normalizados para calcular la extensión. Se emplean la siguientes definiciones:

EFP Puntos Funcionales de la extensión.
ADD Puntos Funcionales, sin ajustar, que se deben agregar.
CHGA Puntos Funcionales, sin ajustar, de la parte que se modificó.
CHGB Puntos Funcionales, sin ajustar, antes de la modificación.
DEL Puntos Funcionales, sin ajustar, que se quitan del proyecto.
CFP Puntos Funcionales para realizar la conversión de datos para el proyecto nuevo.
VAFA factor de ajuste de los Puntos Funcionales después de realizada la extensión del proyecto.
VAFB factor de ajuste de los Puntos Funcionales en el proyecto viejo.

La ecuación recomendada por el IFPUG para calcular el **incremento** de Puntos Funcionales es:

$$EFP = (ADD + CHGA + CFP).VAFA - (CHGB + DEL).VAFB$$

En casos que se cambie de tecnología el problema es similar a una extensión de proyecto. Es importante tener en cuenta el **cambio de los factores de ajuste**, cosa que es seguro que ocurrirá, así como las extensiones y ampliaciones que son casi inevitables. En todo problema de emigración de sistemas aumenta lo que se espera del sistema, simplemente por el hecho que la tecnología de la información tiene prestaciones crecientes. Este es un punto que debe tener siempre en cuenta.

9.3 Estudio de casos de proyectos parciales

Caso 20: Extensión de un proyecto existente

Un sistema de información posee 4.100 Puntos Funcionales. Fue adquirido hace algunos años y se desea emigrarlo de tecnología, mejorar sus condiciones de trabajo y realizar algunos ajustes en la programación. El relevamiento preliminar muestra que se debe:

- Descartar un 30% del sistema porque no se lo usa.

- Realizar cambios en un 30% del sistema porque hay cambios en las necesidades de información. Estos cambios se estima que significarán un aumento de un 50% en sus prestaciones.
- Agregar un 20% de funcionalidades nuevas.
- El sistema original tenía un coeficiente de ajuste de 0,95 y se estima que el nuevo sistema, luego de la emigración en tecnología, tendrá un factor de 1,15.

Con esta información se pueden estimar los diferentes elementos de la ecuación de cambio. Es importante observar que los Puntos Funcionales sin ajustar del sistema original son 4316.

ADD	Puntos Funcionales a agregar	20%	863
CHGB	Puntos Funcionales antes de la modificación	30%	1295
CHGA	Puntos Funcionales luego de la modificación		1942
DEL	Puntos Funcionales que se quitan	30%	1295
CFP	Puntos Funcionales para conversión de datos		0

VAFA	coeficiente de ajuste luego de la modificación	0,95
VAFB	coeficiente de ajuste antes de la modificación	1,15

Con estos valores, la ecuación de cálculo del incremento de los Puntos Funcionales del sistema modificado es:

$$1{,}15 \; (\; 863 + 1942 \;) - 0{,}95 \; (\; 1295 + 1295 \;) = 733$$

Esta será la cifra que permite estimar el tamaño de la extensión. Una vez realizada, el sistema final tendrá:

$$1{,}15 \times 4316 + 733 = 5696$$

Puntos Funcionales ajustados.

10. EXTENSIONES DE LA METODOLOGÍA

10.1 Introducción

Es frecuente que se considere que la metodología de los Puntos Funcionales es incompleta. En algunos casos esta idea es falsa y proviene de una falta de experiencia. En otros casos es justificada.

En este capítulo presentamos algunas extensiones propuestas a la metodología, con la finalidad de extender los campos de aplicación o los alcances conceptuales.

10.2 Metodología Mark II (Symons)

Charles Symons, en Inglaterra, 1988 propuso una revisión de la metodología de Albrecht. Los cuatro puntos mayores de su modificación eran:

- Emplear **entidades y relaciones** en lugar de archivos lógicos.
- Modificar la metodología de contar de modo de **cumplir con el principio de superposición**: un sistema compuesto debe dar lo mismo que la suma de sus partes.
- Convertir la metodología en una técnica para estimar el **esfuerzo de proyecto**.
- Agregar **seis** modificadores adicionales.

Sin duda las propuestas poseen diferente importancia y justificación. En lo que sigue intentaremos dar una visión crítica de la metodología Mark II

Cambiar la noción de archivos lógicos por entidades y relaciones es un punto de vista moderno y que encuentra gran aceptación. Sin duda la difusión de herramientas CASE y el empleo casi universal de manejadores

de base de datos hacen que esta extensión sea necesaria. El tema está en estudio en el IFPUG y es posible que ocurra una revisión de la metodología en el futuro. Es importante destacar, sin embargo, que las extensiones deben mantener **compatibilidad** con las medidas del pasado, de otro modo se perderá algo muy importante.

Modificar la manera de contar de modo que se cumpla el principio de superposición, en opinión de este autor, es contrario a la idea de la **buena estructuración** y a la idea de complejidad misma. No parece plausible que se retroceda en este punto, especialmente puesto que fue el propio IFPUG quien introdujo este criterio, en contra de las prácticas de Albrecht.

Convertir la metodología de los Puntos Funcionales en una técnica de cálculo de esfuerzo es un error conceptual. Significa regresar a los planteos del pasado cuando no se diferenciaba entre métricas y ecuaciones empíricas de predicción de esfuerzo. Por otra parte, en tanto que los Puntos Funcionales pretender ser independientes de la tecnología (cosa que no es más que una aspiración), la predicción del esfuerzo claramente depende de la tecnología. Este punto parece totalmente inaceptable.

El agregado de modificadores nuevos es un hecho natural y debe aceptarse como tal. La manera de conseguir independencia de la tecnología tal vez se encuentre en esta posibilidad. En la opinión personal de este autor, siempre debería ser posible agregar modificadores, la metodología no está cerrada a este planteo. La cuestión es si se deben agregar seis o cien y cuáles son estos modificadores. A los efectos de ilustrar este punto basta pensar en el modificador por las entradas on–line. Es claro que con el pasaje del tiempo, este modificador ha perdido importancia. Tal vez ha llegado la hora de quitarlo y reemplazarlo por algún otro modificador más relevante con las tecnologías actuales y que se relacione, por ejemplo, con el manejo gráfico o con el empleo de objetos.

10.3 Los nuevos modificadores

La propuesta de Symons comprende seis nuevos modificadores:

- Existencia de grandes interfaces de comunicación con otros sistemas.

- Existencia de condiciones de seguridad.
- Programación a la cual acceden terceras partes.
- Exigencias especiales de documentación.
- Existencia de un entrenamiento especial del personal.
- Existencia de un hardware especial.

Todos los modificadores propuestos tienen sentido, pero solamente interesan en proyectos grandes, de cierto tipo. En los hechos puede pensarse que estos modificadores se han incorporado a la tecnología pero que, en la mayoría de los casos, su nivel es cero. No modifican el factor de ajuste por lo tanto.

10.4 Feature Points (Capers Jones)

Esta extensión de la metodología fue desarrollado por Capers Jones en 1986, de Software Productivity Research, consultora de Estados Unidos [JON91]. La principal carencia que se buscaba corregir era la posibilidad de extender la metodología a los sistemas de Tiempo Real, a las aplicaciones de ingeniería o con una fuerte componente matemática. Sin duda este caso no está dentro de los alcances de la metodología de IFPUG que solamente comprende sistemas de información tradicionales.

La cuenta de Puntos Funcionales en los casos mencionados tiende a ser más baja de lo esperado, como si estos sistemas fuesen más simples de lo esperado. Para corregir este hecho es que se introducen los Feature Points.

Los Feature Points son una extensión, **no compatible en la teoría**, de los Puntos Funcionales. Sin embargo, en la práctica, la metodología es compatible en las áreas de los sistemas de información convencionales, que es el único lugar donde interesa mantener la compatibilidad..

La idea básica de los Feature Points consiste en introducir una sexta funcionalidad: **la complejidad de los algoritmos**. Al introducir una nueva funcionalidad es necesario hacerlo a expensas de las otras cinco funcionalidades, de otra manera habría una importante incompatibilidad.

Para introducir la complejidad de los algoritmos, se disminuye el **peso** de los archivos y de allí los problemas de compatibilidad teórica. Esta extensión

tiene también consecuencias sobre el principio de superposición de sistemas. Por lo demás se emplea la metodología original, calificando las funciones en tres niveles de complejidad.

Para clasificar en niveles la nueva funcionalidad es necesario tener una manera de considerar los algoritmos. Una manera simple consiste en construir cuadros de clasificación para todos los algoritmos de uso corriente.

En una segunda metodología se puede definir el **peso** de un algoritmo para ser usado en el cálculo de los Feature Points mediante una ecuación teórica. Este peso se puede definir mediante un cuadro de doble entrada (una métrica de Halstead) según sea la cantidad de pasos o reglas empleadas y la cantidad de argumentos.

En definitiva, todavía es una metodología en proceso de definición y normalización. No existen indicios que el IFPUG trabaje en esta dirección.

10.5 Esquema de cálculo

A los efectos de ilustrar el mecanismo de cálculo, suponiendo que no se clasifican las funcionalidades en tres niveles (tal como era el cálculo original de Albrecht) se tiene el siguiente esquema:

Cuadro 7: Esquema de cálculo de los Feature Points

Número de algoritmos	x 3 =
Número de entradas	x 4 =
Número de salidas	x 5 =
Número de consultas	x 4 =
Número de archivos internos	x 7 =
Número de interfaces externas	x 7 =
Total sin ajustar	suma
Coeficiente de ajuste	
Feature Points ajustados	factor

Como podemos apreciar se “desvían” tres puntos desde los archivos hacia la cantidad de algoritmos y aquí ocurre la modificación más importante respecto a la metodología.

Capers Jones ha estudiado la relación Feature Points/Function Points sobre casos reales. Los datos que ha publicado indican las siguiente relaciones [JON91]:

Cuadro 8: Relación entre Feature y Function Points

Sistema de información, batch	0.8
Sistema de información, on–line	1
Manejo de base de datos, on–line	1
Sistemas de conmutación	1.2
Sistemas de Tiempo Real, embedded	1.35
Sistemas de automatización de fábricas	1.5
Sistemas de diagnóstico y predicción	1.75

Como podemos apreciar, la nueva metodología mantiene una compatibilidad práctica importante en los casos de sistemas de información. En cambio, a medida que se consideran sistemas con un peso más importante en los algoritmos, puede llegarse a un 75% adicional del valor si se aplicara (incorrectamente) la simple metodología de los Puntos Funcionales.

11. APLICACIONES DE LAS MÉTRICAS

11.1 Campos de aplicación

Las diversas métricas de proyecto pueden ser aplicadas a muchos campos de acción. Sin ánimo de considerar todos los campos, podemos señalar:

- Medida del tamaño de un proyecto.
- Medida de la productividad en software.
- Medida de la calidad del software.
- Usos económicos de las medidas.
- Análisis de impacto y de atributos.

La medida del tamaño de un proyecto es el objetivo principal de toda métrica, no merece por lo tanto más comentarios. Las restantes aplicaciones merecen ser consideradas con un poco de atención.

11.2 Tamaños típicos de proyectos

A los efectos de realizar comparaciones es útil tener una idea de los tamaños involucrados en algunos proyectos típicos. Los ejemplos presentados son cifras tomadas de Capers Jones en Puntos Funcionales. A los efectos de tener idea de la dispersión de resultados, se indican intervalos típicos que encierran la mayoría de los casos. [JON91]

Cuadro 9: Tamaños de proyectos típicos (Puntos Funcionales)

Procesador de palabras	500	3.500
Compilador	600	2.000
Herramienta CASE	1.000	15.000
Planilla electrónica	1.000	3.000
Sistema operativo	5.000	50.000
Sistema de reservas aéreas	25.000	50.000

Estas cifras muestran que los sistemas pueden ser clasificados en tres zonas:

- Un sistema con algunos cientos de Puntos Funcionales puede ser considerado un **proyecto pequeño**.

- Un sistema de algunos miles de Puntos Funcionales representa ya un proyecto de dimensiones importantes, lo que hemos llamado **proyecto mediano**.

- Los **proyectos grandes** están en la zona de las decenas de miles de Puntos Funcionales.

11.3 Evolución histórica de la productividad

La información del cuadro que sigue es una estimación de Capers Jones de la productividad en los Estados Unidos a lo largo del tiempo. La unidad de medida es Puntos Funcionales por mes–persona en las tres áreas típicas de la programación. [JON91]

Cuadro 10: Productividad a lo largo del tiempo

Década considerada	Sistemas de gestión de información	Programación de base	Programación para uso Militar
1950	1,0	0,5	0,8
1960	2,0	1,0	1,5
1970	4,0	2,0	2,5
1980	6,0	3,0	3,5
1990	8,0	4,0	5,0
2000	12,0	7,0	9,0

Es interesante observar el aumento sistemático de la productividad en el desarrollo de la programación. Este aumento obedece a la mejora constante de las herramientas empleadas. En la década del 50 solamente existía el ASSEMBLER como lenguaje de trabajo. En la década del 60 la aparición de los lenguajes de alto nivel (FORTRAN y COBOL) duplicó la productividad promedio. Esta tendencia continuó la década siguiente, en la cual se

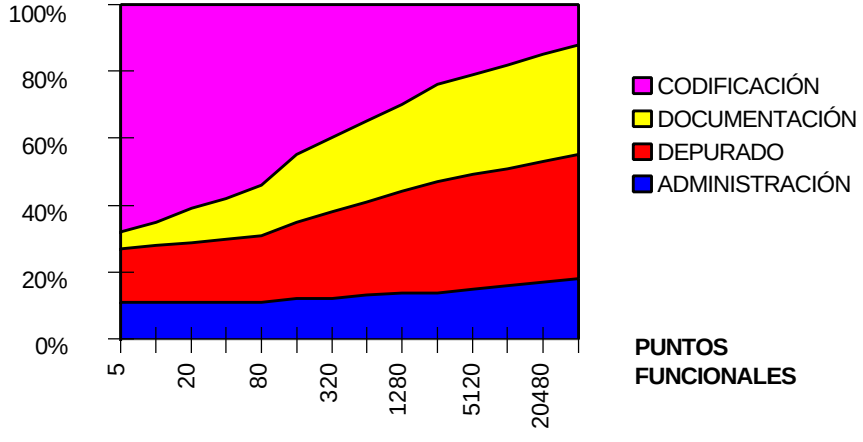
abandonó completamente el ASSEMBLER en los sistemas de información. Un nuevo aumento de productividad ocurre en las dos siguientes décadas con la introducción de los manejadores de base de datos, los lenguajes de cuarta generación y las herramientas CASE.

También es interesante observar que la programación de base tiene una productividad efectiva del orden de la mitad de la encontrada en los sistemas de información. La programación para uso militar se encuentra a mitad de camino entre ambas.

Estas cifras sirven como indicación de la tendencia histórica pero no como herramienta para ingeniería inversa. Es un error muy grave considerar, por ejemplo, que la productividad actual se pueda tomar como algún valor entre 8 y 12 Puntos Funcionales por mes–persona. Hacer esto significa desconocer todo cuanto se ha presentado acerca de la gestión de proyectos.

11.4 Distribución del esfuerzo de un proyecto

La distribución del esfuerzo en un proyecto en sus diferentes rubros depende del tamaño del proyecto. La curva que sigue tiene en cuenta cuatro áreas de actividad: el análisis y la codificación; la documentación; el depurado y la administración del proyecto. [JON91]:



Esta distribución del esfuerzo está hecha para proyectos en COBOL mayoritariamente. Esto hace que debamos considerar, por ejemplo, en 180 Puntos Funcionales la frontera de los proyectos medianos y en 2.500 la frontera de los proyectos grandes. No obstante esto, los resultados tienen un carácter general.

Puede apreciarse que para todos los tamaños de proyecto, el esfuerzo de administración es razonablemente constante y del orden del 10% del esfuerzo total.

Por el contrario, la magnitud más variable es el esfuerzo de análisis y codificación del proyecto. Para proyectos pequeños puede ser el 65% del esfuerzo total. En el otro extremo, en proyectos grandes, es **solamente un 20%**. Este hecho refleja uno de los puntos esenciales de la deseconomía de escala. Al mismo tiempo, esta variación muestra que en los proyectos grandes las herramientas CASE no tienen demasiado campo de acción. Desde el momento que la mayor parte del esfuerzo del proyecto se encuentra en la documentación y el depurado, no inciden sino indirectamente en la mejora de la productividad.

El esfuerzo de depurado y documentación no tiene variaciones según el tamaño de los proyectos. La documentación varía desde un 5% en los proyectos pequeños a un 40% en los proyectos grandes. El depurado cuenta un 20% en los proyectos pequeños y aumenta a un 30% en los proyectos grandes.

A los efectos prácticos se puede tomar el siguiente cuadro para **estimar** estas cuatro componentes:

Cuadro 11: Distribución del esfuerzo según el tamaño del proyecto

	proyecto pequeño	proyecto mediano	proyecto grande
Análisis y codificación	65%	45%	20%
Documentación	5%	20%	40%
Depurado	20%	25%	30%
Administración	10%	10%	10%

Los valores indicados son solamente una referencia que pueden ser usados en ausencia de indicaciones mejores. Según sea el proyecto, además, podrá corregirse algo en los casos que se sepa que existe una desviación importante de algunos de estos parámetros, por ejemplo, si se exige una alta confiabilidad o una excelente documentación.

En los casos de estudio se ilustra el empleo de este cuadro.

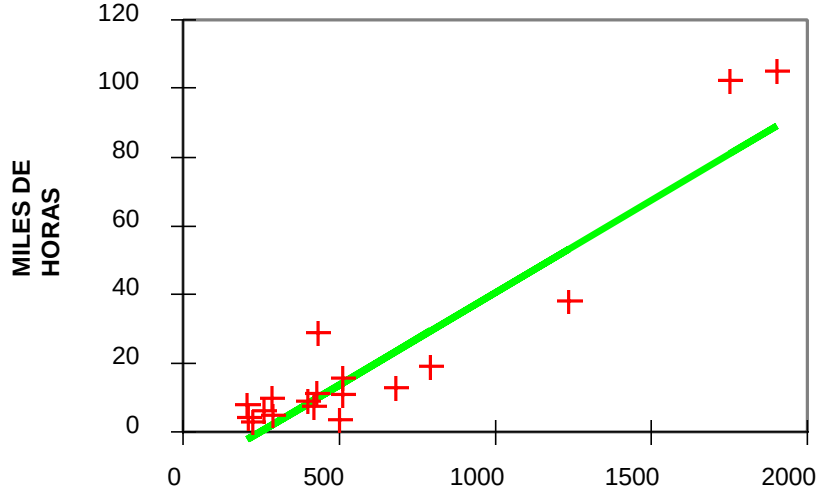
11.5 Parámetros económicos de un proyecto

Sin duda la aplicación más importante de las métricas se refiere a la estimación de los parámetros económicos de un proyecto: esfuerzo, tiempo de entrega y número de personas.

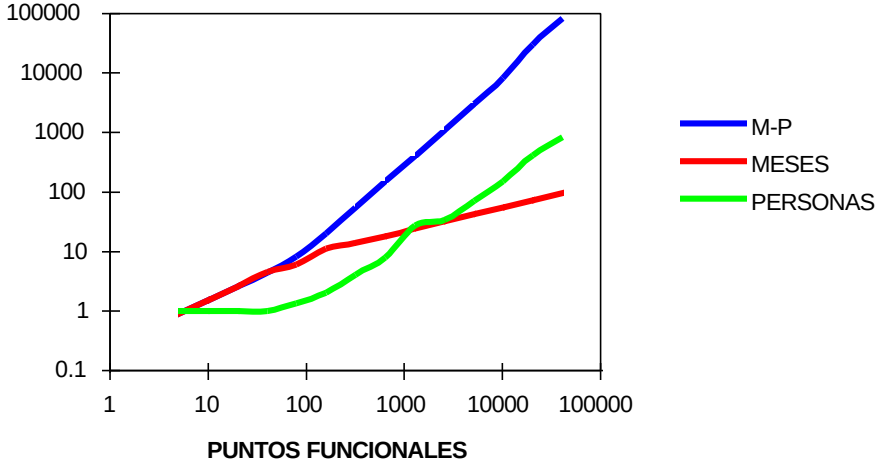
En sus orígenes, la metodología de los Puntos Funcionales fue también una metodología de estimación de esfuerzo. Uno de los primeros resultados de Albrecht fue relacionar sus medidas (que correspondían a proyectos realizados en COBOL mayoritariamente) con el esfuerzo de realización del proyecto. La ecuación empírica que proponía era [ALB83]:

54 . PF – 13390 (horas de trabajo)

Esta ecuación se fundamentaba en una colección de proyectos que presentaba el siguiente aspecto:



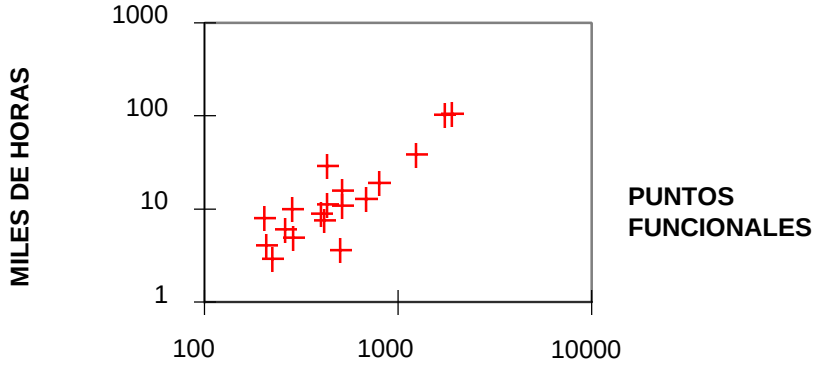
Vale la pena recordar que este estudio es de fines de la década del 70 porque hoy muy poca gente intentaría realizar un **ajuste lineal** de los datos de esfuerzo de un proyecto. Así por ejemplo, cuando Capers Jones presenta datos empíricos de proyectos emplea un diagrama doblemente logarítmico [JON91]:



El diagrama de Jones (excepto por el tramo de la curva de personas que no considera fracciones de persona) muestra una conducta de tipo potencial que se manifiesta por tramos rectos. Se puede advertir también que existen

quiebres cerca de 100 (frontera de proyecto pequeño) y cerca de 2.000 (frontera de proyecto grande) Puntos Funcionales.

Esta curvas justifican una revisión de los datos de Albrecht. Pasando sus datos a un diagrama logarítmico, el aspecto cambia completamente:



En este diagrama se puede advertir una clara alineación de los proyectos que evidencia también una conducta de tipo potencial.

No cabe duda hoy que estas curvas dependen fuertemente de la tecnología empleada. Los datos medidos para proyectos en COBOL no pueden ser usados para otro lenguaje de programación. Se debe ser muy cauteloso al realizar este tipo de extrapolaciones, tal como se lo muestra en los casos de estudio.

11.6 Lenguaje y Puntos Funcionales

En muchas oportunidades es necesario vincular Puntos Funcionales con cantidad de líneas de código. Un ejemplo típico es en los casos de emigración de tecnología.

Por cierto que no existe ninguna vinculación simple entre dos conceptos tan diferentes. No obstante esto, es posible obtener una primera aproximación mediante una tabla empírica realizada por Capers Jones [JON91] y que se presenta en el cuadro:

Cuadro 12: Relación entre líneas de código y Puntos Funcionales

Lenguaje	Líneas por PF
ASSEMBLER	320
C	128
FORTRAN 77	105
ANSI COBOL 74	105
ADA	71
PROLOG	64
manejador de base de datos	40
lenguaje de cuarta generación	20
generador de programas (CASE)	16
macros de una planilla electrónica	6

Las cifras de este cuadro permiten realizar una primera aproximación de gran utilidad, pero no pueden reemplazar nunca la medida directa. Queda claro, también, que no es posible aspirar a tener una única ecuación que relacione Puntos Funcionales con el tamaño de un proyecto.

11.7 Medida de la calidad del software

En el área de la medida de la calidad, las métricas pueden ser una excelente ayuda. Así por ejemplo, son interesantes algunos indicadores tales como:

- Cantidad de cambios en un sistema, por Punto Funcional.
- Cantidad de errores por Punto Funcional.
- Fallas de la aplicación por Punto Funcional.

En el campo de la medida de defectos, si bien no se tiene todavía una interpretación general, existen algunos resultados conocidos que son de utilidad.

En el cuadro que sigue se presentan la cantidad de bugs encontrados según el tipo y el momento de ocurrir para una aplicación en COBOL de, aproximadamente, 50.000 líneas y 500 Puntos Funcionales. [JON91]

Cuadro 13: Cantidad de errores en un sistema

FASE DEL TRABAJO	errores por KLOC	errores por PF
Revisión de especificación	2.5	0.25
Revisión del diseño	5.0	0.50
Inspección del código	10.0	1.00
Pruebas de integración	3.0	0.30
Pruebas de aceptación	2.0	0.20
Encontrados en el primer año	8.0	0.80

Según estas cifras, en un proyecto de 50.000 líneas, en el caso de buena calidad, hay 250 defectos que serán detectados en el primer año. Esto supone que prácticamente **cada día de uso**, durante todo el primer año, se encontrará un defecto nuevo!

11.8 Usos económicos de las medidas

Una métrica tiene gran cantidad de aplicaciones de tipo económico. En la medida que la métrica está normalizada, como es el caso de los Puntos Funcionales, sus posibilidades se multiplican. Por el sólo hecho que es posible acudir a un árbitro independiente que pueda medir (y obtener medidas muy próximas a las realizadas por otros técnicos) es una valiosa herramienta para las relaciones comerciales.

Antes de comenzar a exponer las virtudes, es necesario comenzar con una advertencia de carácter general. **Las métricas no sirven para la medida de la productividad individual de un trabajador**. En la medida que un proyecto informático es obra de un equipo, conduce a resultados absurdos intentar medir la productividad individual. Así por ejemplo, el equipo de documentación y el de depuración no produce Puntos Funcionales y, sin embargo, realiza un aporte esencial para el desarrollo del proyecto.

Un primer grupo de aplicaciones de tipo económico se encuentran relacionadas con los análisis de costo de proyectos, antes de su ejecución. Una métrica, complementada con sus ecuaciones empíricas de esfuerzo, personas y tiempo permite:

- Realizar el análisis preliminar de un proyecto.

- Estimar tamaños sobre la base de un anteproyecto que permita una buena medida inicial.
- Estudiar la factibilidad económica de un proyecto.
- Estimar el costo final de un proyecto.

En el caso de una métrica normalizada, se puede agregar a este hecho, la posibilidad de disponer de una paramétrica para contratos de obra. En muchos casos pueden existir cambios durante la marcha del proyecto. Una métrica como los Puntos Funcionales permite proponer, antes de comenzar, una ecuación paramétrica de ajuste de precios por cambios en más o en menos. La importancia de esta propuesta es que sus resultados pueden ser sometidos a un árbitro, en el caso de una situación de conflicto.

En otro orden de cosas, una métrica permite realizar comparaciones sobre la base de mediciones. Así por ejemplo, se pude:

- Realizar un análisis de impacto de un parámetro.
- Comparar atributos de una herramienta o una solución.
- Medir las modificaciones que introduce una nueva tecnología.
- Comparar alternativas tecnológicas.
- Comparar alternativas de diseño.

Las métricas permiten una organización racional del trabajo informático. Por ejemplo, son interesantes los indicadores:

- Esfuerzo de mantenimiento por punto funcional.
- Carga de mantenimiento a cada persona.

La métricas permiten obtener indicadores de significado estadístico o de control, tales como:

- Costo de proyecto por Punto Funcional.
- Costo de mantenimiento por Punto Funcional.
- Costo de la empresa por Punto Funcional.
- Estimación del valor total del portafolio informático de una empresa.
- Seguimiento de la tendencia anual del portafolio de una empresa.

- Seguimiento de los Puntos Funcionales entregados o incorporados y su tendencia anual.
- Productividad global de una empresa productora de programación.

Estos indicadores tienen poca importancia técnica, pero en algunos casos pueden servir como indicadores globales de gestión. Es muy difícil realizar comparaciones con estos indicadores. Tomemos el primero, por ejemplo: no existe nada parecido al "precio" o al "costo" de un Punto Funcional. Debido a la deseconomía de escala, cuanto mayor sea el proyecto, más caro será cada uno de sus Puntos Funcionales.

El portafolio informático de una empresa es un indicador interesante. Da una idea de la importancia de sus sistemas de información y es una medida indirecta de su valor. En el cuadro siguiente se presentan portafolios típicos, en Puntos Funcionales, para algunos sectores de la industria norteamericana. [JON91]

Cuadro 14: Portafolios típicos de algunas empresas

Banco pequeño (norteamericano)	125.000
Compañía manufacturera mediana	200.000
Compañía grande de teléfonos	450.000
Gran banco internacional	450.000
Gran fabricante de computadoras	1:650.000

También es un dato curioso el tamaño de la programación usada por cada trabajador en diferentes sectores (Estados Unidos). [JON91]

Cuadro 15: Tamaño de la programación usada por trabajador

Grupo ocupacional	PF
Agente de viajes	35.000
Funcionario de reserva de pasajes	30.000
Ingeniero eléctrico o electrónico	25.000
Ingeniero de software	15.000
Gerente de proyecto de software	3.500

Estos valores, si bien no tienen ninguna aplicación directa, dan idea del grado de informatización de diferentes sectores ocupacionales. Sin duda el manejo de los sistemas de reserva de pasajes convierte al sector turismo en uno de los más automatizados de todas las actividades. Estas cifras contrastan mucho con las herramientas para la gestión de proyectos de software.

12. PROBLEMAS DE INGENIERÍA INVERSA

12.1 Heritage software

El problema de la conversión de viejos sistemas de información, que se encuentran operativos y que han trabajado por años (heritage software), es un desafío formidable. Para esto se han desarrollado múltiples técnicas de ingeniería del software. No es nuestro propósito analizarlas aquí.

El punto que nos interesa particularmente es analizar el heritage software mediante herramientas a los efectos de extraer información útil para la emigración de tecnología y la actualización de funcionalidades.

Los viejos sistemas, si bien no se pueden mantener, ejecutan en plataformas obsoletas, están mal documentados, sus prestaciones son anticuadas e insuficientes, poseen una información muy útil que puede ser utilizada en el futuro.

Es por esta razón que nos ocuparemos especialmente de este tema.

12.2 Relevamiento por número de líneas

En la medida que el heritage software está realizado en un lenguaje con líneas de código (desde COBOL hasta un lenguaje de cuarta generación, pasando por RPG), estas líneas de código tienen una información preciosa que es necesario relevar.

La primera tarea del analista será entonces, tomar todos los módulos del heritage software y contar el número de líneas que poseen. Esta tarea puede ser muy sencilla si se cuenta con utilitarios para contar líneas. El caso típico es en UNIX donde hay una herramienta clásica para esta tarea. En otras plataformas habrá que recurrir a los editores de programas o a otros recursos que permitan contar el número de líneas.

Si se conoce la cantidad de módulos y el lenguaje empleado se pueden emplear los datos del Cuadro 2. También se puede emplear la metodología de COCOMO en sentido inverso: si se conoce o se puede estimar el esfuerzo empleado en el desarrollo de un sistema, se puede estimar las KLOC que el sistema posee. Esta cifra es global, pero es mejor que nada.

Una vez realizado este relevamiento, es conveniente armar una planilla electrónica con el nombre de cada módulo, el sistema al cual pertenece, el lenguaje empleado y el número de líneas del módulo. En este proceso es necesario dejar de lado la programación transitoria, la programación fuera de uso: solamente deben quedar los módulos finales efectivos de cada sistema de información.

El paso siguiente consiste en analizar el porcentaje de líneas vacías o de comentario que tiene. A estos efectos conviene proceder por muestreo. Una muestra adecuada se analiza directamente y se cuentan la cantidad de comentarios encontrada. Como guía, vale la pena señalar que es difícil que haya menos del **10% entre líneas de comentarios y vacías**. En este proceso de análisis es conveniente separar estilos diferentes que pueden corresponder a grupos de trabajo o épocas diferentes.

Llegados a este punto, se puede completar la planilla con el coeficiente que convierte líneas totales en líneas efectivas finales de código.

El último paso de este análisis es convertir líneas de código en Puntos Funcionales. Para esto se emplean técnicas que pueden ser complementarias:

- Se usan los datos del Cuadro 12 o la planilla GPS para realizar la conversión.
- Se elige una muestra conveniente de módulos y en ellos se hace la **cuenta directa** de Puntos Funcionales, tal como se ve en lo que sigue.

El resultado del análisis de líneas de código es una medida, módulo por módulo y sistema por sistema de la cantidad de Puntos Funcionales. Es interesante especular acerca de si esta cuenta está sin ajustar o es ajustada.

En general, los puntos obtenidos por estas técnicas son **sin ajustar** y deben ser ajustados con los coeficientes correspondientes.

12.3 Relevamiento funcional

A partir de la documentación existente, de los sistemas reales o del análisis de los programas fuente, siempre es posible realizar el relevamiento funcional del heritage software. Para esto basta con identificar los archivos e interfaces, entradas, salidas y consultas.

El proceso de relevamiento de un sistema existente es simple y directo y es la fuente más precisa de todas. En particular, este método es altamente recomendable para **calibrar** la equivalencia entre líneas de código y Puntos Funcionales: los diferentes estilos de programación hacen que los resultados del Cuadro 12 sean solamente una primera aproximación.

12.4 Relevamiento por objetos

El relevamiento por objetos es una de las técnicas más interesantes de análisis para el heritage software. En la medida que los sistemas fueron desarrollados con herramientas de mayor nivel que el COBOL, en alguna medida se puede aplicar el concepto de objetos.

En este contexto se llama objeto a cualquier estructura de programación identificada como diferente de las demás. Archivos, bases de datos, procedimientos, diseños de pantalla, reportes automáticos, listados, programas utilitarios, todos estos elementos pueden aparecen en el heritage software.

La primera etapa consiste en realizar un relevamiento de todos los objetos que se pueden reconocer. La segunda etapa consiste en armar una planilla, sistema por sistema, de los objetos que posee, tal vez diferenciándolos en tres niveles: simple, promedio y complejos. La tercera etapa del análisis consiste en asignar una cuenta de Puntos Funcionales a cada objeto. Esta asignación se hace luego de una evaluación teórica o práctica.

Como resultado de todo esto se dispone de una planilla de análisis que permite calcular los Puntos Funcionales (posiblemente sin ajustar) de los sistemas en estudios.

12.5 Comparación entre diferentes valores

Es normal que en el proceso de análisis de heritage software se disponga de más de una estimación para el sistema, especialmente si se aplicó más de una metodología.

No hay nada mejor que disponer de más de una estimación para el mismo sistema. Es seguro que en una primera instancia las estimaciones **no coinciden**. Se inicia entonces una etapa de conciliación de resultados.

El proceso de conciliación de resultados consiste en revisar toda las estimaciones realizadas otra vez, cotejando una con la otra, para detectar criterios dispares, desviaciones de parámetros, errores de estimación, módulos sobrantes o faltantes y toda otra causa similar de error.

Se debe insistir tantas veces como sea necesario en el proceso de conciliación hasta tener una coincidencia al menos **dentro de un 20% de error**. Llegados a este punto se puede considerar que el valor promedio es una buena estimación de los Puntos Funcionales sin ajustar del heritage software.

12.6 Ajuste de resultados

Al igual que en el caso normal, se debe proceder ahora a realizar el ajuste. Para esto basta con repasar los 14 modificadores y asignarles valores **en las condiciones en que fue realizado el heritage software**.

Una vez en posesión de los Puntos Funcionales ajustados, es prudente realizar un contraste final con valores típicos de otras instalaciones a los efectos de realizar una última comprobación que permita confiar en los valores asignados al heritage software.

12.7 Uso de la planilla GPS

La planilla GPS suministra una función útil para el análisis del heritage software. La función tiene el aspecto:

PF(Lineas, Lenguaje)

Donde el argumento **Líneas** es la cantidad final, efectiva, de líneas de código empleadas en un módulo, un programa o un sistema. La variable **Lenguaje** indica el lenguaje (en mayúsculas) que se considera. Si se emplea un lenguaje no válido, o escrito en una forma no debida, se devuelve cero.

12.8 Estudio de casos

Caso 21: Replanteo de un heritage software grande

Se trata de estudiar un sistema de información de producción de fábrica, escrito en RPG II para un IBM /36 y luego emigrado a un IBM AS/400. Como existen planes de cambio de tecnología, a los efectos del estudio económico se desea tener una buena estimación de la cantidad de Puntos Funcionales que el sistema posee.

Para obtener esta medida se intentan dos caminos: el estudio de la programación existente y el análisis directo de las funcionalidades del sistema.

El examen de la programación permitió construir el siguiente cuadro. Se consideraron todos los objetos y se asignó, según fuese el caso, una cantidad de Puntos Funcionales. Cada DFU se lo consideró una entrada simple, cada QRY como una consulta simple. Los programas RPG II se evaluaron por su cantidad de líneas.

tipo	cantidad	líneas	PF	comentarios
DFU36	109	713	33	entrada de datos simple
DSPF36	342	23733	0	pantallas
MNU36	87	2486	0	menú
OCL36	1046	29231	0	archivo de comandos

QRY36	158	1010	632	query sobre una base
RPG36	772	216151	2702	fuente RPG
UNS36	621	81065	1013	fuente RPG
TOTAL de Puntos Funcionales			4380	

La evaluación (simplificada) de las funcionalidades se hizo mediante el análisis directo de archivos, salidas y pantallas. Para realizar este análisis de realizó una depuración previa para evitar considerar objetos transitorios que podrían distorsionar la medida. Las consultas y los DFU completaron el siguiente cuadro:

archivos	162	medida directa
salidas	169	medida directa
consultas	158	query
entradas	173	pantallas menos salidas
entradas	11	DFU

La clasificación en tres niveles condujo a la estimación de los Puntos Funcionales:

	simples	promedio	complejas	PF
entradas	132	52		604
salidas	276	118		1697
consultas	158			632
archivos	110	52		1290
interfaces				0
TOTAL				4222

Existe un acuerdo excelente entre las dos estimaciones. No puede considerarse importante la diferencia entre 4222 obtenido por el análisis de funcionalidades y 4380 por tamaño de programas, lo cual supone un error menor que el 4%. Vale la pena observar que los elementos que se tienen en cuenta para una y otra estimación son diferentes, de modo que la coherencia de resultados es sumamente importante.

Caso 22: Estimación de Puntos Funcionales a partir de líneas de COBOL

Consideremos el Caso 9, un proyecto grande realizado en COBOL. Es posible realizar la estimación de Puntos Funcionales de dos maneras diferentes.

En el Caso 12 se estimaba, por funcionalidades, el intervalo de valores extremos de 3735 a 7565 Puntos Funcionales. Se tomaba como valor probable (con la hipótesis de distribución 80–20 entre simples, promedio y complejos) la cantidad de 4110 puntos.

En este proyecto se tenían 732.538 líneas de COBOL. Tomando 105 líneas de código por Punto Funcional, el valor estimado para estos sistemas es 6976 puntos.

Como puede apreciarse, las dos estimaciones discrepan, pero se encuentran dentro del intervalo de valores extremos. Solamente un análisis más detallado, que permita clasificar las funcionalidades en tres niveles permitirá afinar la cuenta del sistema.

Caso 23: Estimación de esfuerzo de un proyecto COBOL

Tomemos la estimación del Caso 9 realizada mediante COCOMO. Disponemos de otras maneras de realizar la estimación del esfuerzo.

Mediante la ecuación de Albrecht presentada en la sección 11.5, si tomamos los valores extremos de Puntos Funcionales del Caso 22 pueden estimarse las horas de trabajo. Lo mismo puede hacerse con la función COBOL de la planilla GPS. Con la media de 152 horas por mes–persona, el esfuerzo se puede convertir en meses–persona. El resultado es el siguiente cuadro comparativo:

COCOMO	732,5	KLOC COBOL	2689
Albrecht	3735	Puntos Funcionales	1238
Albrecht	7565	Puntos Funcionales	2599
GPS	3735	Puntos Funcionales	965
GPS	7565	Puntos Funcionales	2251

De esta comparación puede apreciarse que es importante afinar la cuenta de los Puntos Funcionales para disminuir el intervalo de valores. Se observa que la estimación de COCOMO es mayor que las otras. También es visible que el efecto deseconomía de escala está presente en la función COBOL de GPS en tanto que no lo está en la ecuación de Albrecht. Finalmente, las cifras son perfectamente compatibles con las indicadas por Jones en la figura de la sección 11.5.

13. LAS HERRAMIENTAS CASE

13.1 La dificultad de las herramientas CASE

Las herramientas CASE (con todo su espectro de casos) suelen presentar desafíos importantes para la gestión de proyectos de software. Por un lado son herramientas de diseño y programación de alta eficiencia, pero por otro lado pueden no ser aplicables algunos de los conceptos clásicos.

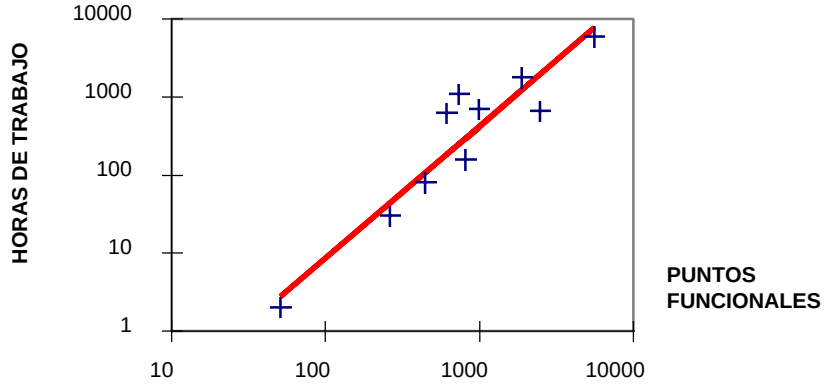
El primer concepto que no es aplicable a las herramientas CASE (lo mismo que a todas los lenguajes de tipo visual) es la cuenta de líneas de código. En la mayoría de los casos el analista o el programador no trabaja con líneas de código o lo hace solamente una pequeña fracción de tiempo. Muchas herramientas son capaces de dar una **imagen en código**, pero este código no fue creado por el equipo de trabajo sino generado automáticamente: no sirve para establecer una métrica a menos que se realice un estudio de validación correspondiente.

Una métrica como los Puntos Funcionales conserva sus posibilidades de aplicación con herramientas CASE pero, de todas maneras, puede ser necesario investigar la validez de su aplicación y se debe investigar a la búsqueda de ecuaciones empíricas aplicables.

13.2 Estudio empírico para GeneXus

GeneXus es una típica herramienta CASE, desde el punto de vista que nos ocupa. A los efectos de establecer una métrica para proyectos es interesante realizar un estudio empírico sobre proyectos finalizados sobre los cuales se tiene información precisa.

La primera idea que se puede explorar es buscar una ecuación empírica del esfuerzo basada en una conducta de tipo potencial que vincule Puntos Funcionales y esfuerzo de proyecto. En la figura que sigue se hace esto:



Como puede apreciarse en el diagrama doblemente logarítmico, existe una excelente correlación entre el esfuerzo y los Puntos Funcionales del proyecto. De aquí resulta una ecuación empírica de esfuerzo.

En los casos considerados se medía el esfuerzo de relevamiento, diseño, implementación y puesta en marcha. No se tenía en cuenta la documentación, la capacitación de personal u otros elementos de proyectos grandes.

13.3 Los modificadores de ambiente

Como es natural, los Puntos Funcionales a través de una ecuación empírica no pueden dar sino un esfuerzo nominal que es necesario ajustar mediante modificadores que se vinculan con el ambiente en el cual se realiza el proyecto.

Puntos Funcionales miden la complejidad del proyecto pero no tienen en cuenta **el ambiente de desarrollo** en el que se trabaja. Por otra parte, la metodología **COCOMO** tiene en cuenta 10 modificadores de **complejidad** y 5 de **ambiente.** Por esta razón, a los efectos de calcular el esfuerzo de un proyecto, se pueden emplear estos modificadores.

Los modificadores que son directamente aplicables a una herramienta CASE, luego de obtener una ecuación empírica son:

- Experiencia de los analistas (**ACAP**).
- Experiencia en la aplicación (**AEXP**).
- Experiencia en la herramienta (**PCAP**).
- Experiencia en la plataforma (**VEXP**).
- Exigencia de plazo de entrega (**SCED**).

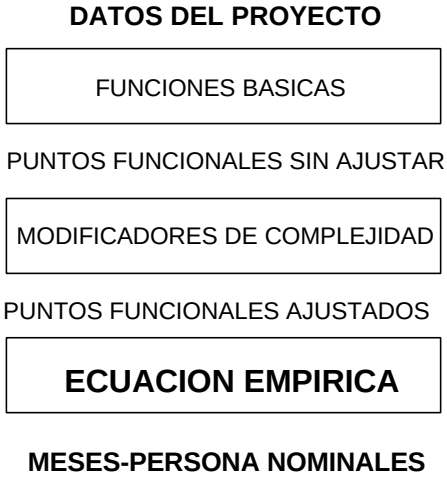
Pero además de estos modificadores, sería interesante disponer de otros modificadores que tuviesen en cuenta otros factores vinculados con el proyecto, tale como:

- Estilo de diseño e implementación del proyecto.
- Complejidad de la documentación.
- Complejidad de la capacitación.
- Cooperación u obstáculo que ofrecerá el usuario.

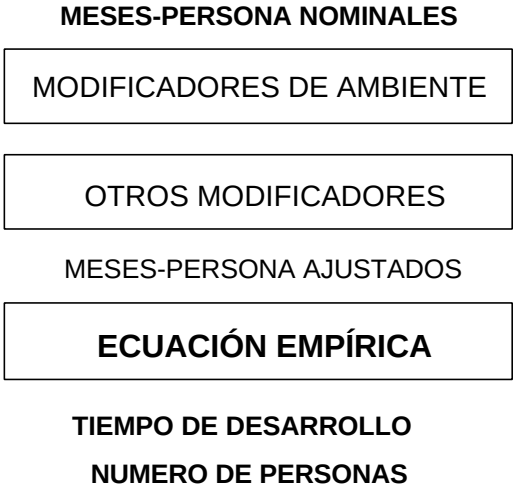
Por cierto que podrían agregarse otros modificadores. Para tratar estos temas o bien se desarrollan técnicas empíricas o bien se puede emplear una estrategia de agregar modificadores a los Puntos Funcionales, tal como propone Symons.

13.4 La metodología GPM

La metodología GPM consiste en aplicar las ideas anteriores a una herramienta CASE. En la primera etapa se determinan los Puntos Funcionales –o una metodología extendida si fuese el caso– del proyecto:



A partir de los Puntos Funcionales, una ecuación empírica (específica para GeneXus) realiza la estimación de los meses–persona nominales. Para corregir el esfuerzo nominal se aplican los modificadores de ambiente –y otros modificadores si fuese el caso– siguiendo la metodología de COCOMO, tal como muestra la figura:



Una nueva ecuación empírica, similar a las empleadas por COCOMO permite calcular tiempo recomendado de desarrollo y, por lo tanto, el número promedio de personas.

13.5 Uso de la planilla GPS

La planilla GPS posee dos funciones que permiten estimar el esfuerzo de un módulo, programa o proyecto a partir de su cuenta de Puntos Funcionales. Las dos funciones son empíricas y dan un resultado nominal que debe ser ajustado, tal como se indicó en este capítulo. El argumento de ambas es **PuntosFuncionales**, valor ajustado a considerar. Las funciones son:

COBOL Esfuerzo nominal en horas para programación COBOL.
GPM Esfuerzo nominal en horas para programación GeneXus.

El uso de estas funciones se analiza en el estudio de casos. Para el cálculo del tiempo recomendado de desarrollo posee otra ecuación empírica que calcula, a partir de los meses–persona, los meses nominales de desarrollo.

TDEV Tiempo nominal de desarrollo, en meses, para un sistema de información que requiere un cierto esfuerzo en meses–persona.

El cálculo de número de personas se realiza mediante los meses–persona de esfuerzo y los meses de desarrollo.

13.6 Estudio de casos en GeneXus

Caso 24: Proyecto máximo que puede realizar una persona

Esta cifra mide bastante bien la eficiencia de la herramienta en la implementación de proyectos.

Para realizar la estimación, se procede por tanteo en la planilla GPS. A 152 horas de trabajo mensuales, este valor en 1300 Puntos Funcionales. Como puede apreciarse, para GeneXus es pequeño (unipersonal) un proyecto que en COBOL sería considerado mediano.

Caso 25: Estimar el esfuerzo de emigración de un sistema grande

Consideremos el proyecto del Caso 21 y tomemos, como media, un valor de 4300 Puntos Funcionales. El análisis de la planilla GPS indica en condiciones medias de implementación y 152 horas de trabajo semanal:

Esfuerzo de desarrollo (meses–persona)	32,3
Tiempo nominal de desarrollo (meses)	9,4
Cantidad promedio de personas	3,4

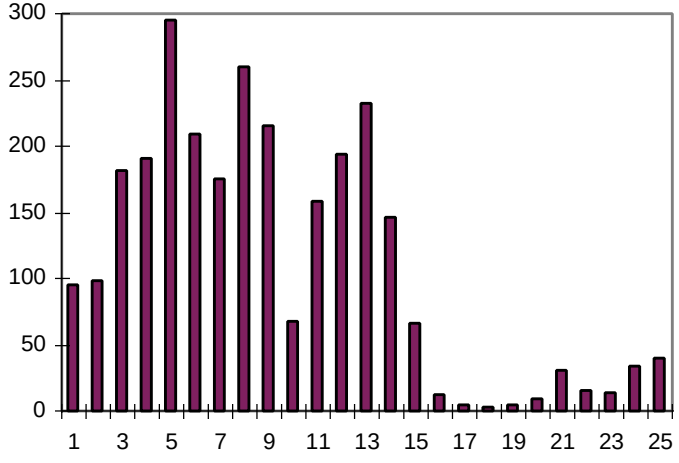
Este proyecto, que originalmente era un proyecto grande, con una herramienta de este poder se convierte en un proyecto mediano.

Vale la pena señalar que si debemos completar el proyecto con capacitación y documentación, es necesario tener en cuenta las cifras del Cuadro 12 para proyectos medianos.

14. EL DESARROLLO EN EL TIEMPO

14.1 Introducción

Un proyecto real no se desarrolla de una manera uniforme en el tiempo. Existen momentos de mucho trabajo y momentos en que el trabajo decae. En general se acepta que todo proyecto posee un pico de dedicación y luego una dedicación descendente hasta su finalización. La figura que sigue presenta horas de trabajo mensuales trabajadas en un proyecto real.



Es posible apreciar una conducta como la señalada. Entre los meses 5 y 8 se advierte un máximo de trabajo. A partir del mes 18 el proyecto parecía finalizado, pero ha sido necesario retomarlo durante los meses 21 a 25 para introducir modificaciones.

Los modelos y las métricas analizadas hasta el momento no consideraban este aspecto del desarrollo de un proyecto, solamente se ocupaban de obtener valores totales para el esfuerzo o el tiempo de entrega. Ahora es necesario precisar estos conceptos.

14.2 El desarrollo en el tiempo de un proyecto

Los problemas dinámicos –o de desarrollo en el tiempo– son la etapa final en el análisis de un proyecto. Esto ocurre tanto en proyectos ya completados como en proyectos nuevos. En el caso de los proyectos nuevos es una instancia que ocurre luego del análisis estático.

Los modelos estáticos suministran cifras globales, los modelos dinámicos analizan la evolución en el tiempo. Son aspectos complementarios del problema. En general, el análisis dinámico ocurre después del análisis estático.

En los hechos, si el proyecto se elabora con suficiente detalle, la metodología clásica de **Pert** y **Gantt** permite estudiar la evolución del proyecto. Para llegar a este punto es necesario un conocimiento con bastante detalle de todos los procesos, etapas, pasos y elementos que comprende el proyecto. Esta herramienta es muy precisa y es un paso obligado del análisis antes de comenzar un proyecto nuevo. Por otra parte, mediante estas herramientas –que se emplean en la gestión de todo tipo de proyectos– permiten detectar puntos críticos, sobrecarga de recursos, camino críticos y otros elementos importantes de conocer antes de comenzar un nuevo proyecto.

Si bien la metodología de Pert y de Gantt es de gran precisión y confiabilidad, a veces no es apropiada para realizar una estimación preliminar rápida o una asignación preliminar de recursos. Por esta razón para la ingeniería del software se han desarrollado herramientas específicas.

El modelo dinámico universalmente empleado en la práctica de la ingeniería del software es el modelo de **Putnam** que obedece a la ecuación de **Rayleigh–Norden.**

Este modelo es un modelo teórico que se ha validado por la experiencia práctica. Es de sencilla aplicación y sus resultados son altamente confiables y útiles en las fases preliminares de elaboración de un proyecto nuevo.

14.3 La fundamentación teórica de Norden

En esta sección presentamos el desarrollo teórico de las ecuaciones. No es necesario seguir este tema para la comprensión de las secciones que siguen y de los casos de aplicación.

Sea ***K*** el esfuerzo total involucrado en un proyecto. Este esfuerzo es la suma –la integral, mejor dicho– de un esfuerzo distribuido en el tiempo. Es el valor que estiman los modelos estáticos.

Sea ***E(t)*** el esfuerzo realizado hasta un instante ***t***. Este esfuerzo proviene de la contribución de las personas involucradas en el proyecto. Sea ***m(t)*** la cantidad de personas que trabajan en el proyecto en este instante. Es claro que se cumple:

$$E(t) = \int_0^t m(t)\ dt$$

$$K = \int_0^\infty m(t)\ dt$$

La hipótesis que realiza Norden es que el número óptimo de personas que deben trabajar en el proyecto, en un instante dado, es proporcional al esfuerzo que queda por realizar. Esta hipótesis es una presunción personal del autor y no es simple de justificar.

De esta hipótesis, derivando, resulta la ecuación diferencial:

$$\frac{d\,E(t)}{dt} = m(t) = a[K - E(t)]$$

En esta ecuación ***a*** es un factor de eficiencia del equipo de proyecto. En general es una función del tiempo, puesto que se puede suponer que existe un efecto de "aprendizaje" durante la realización del proyecto. De acuerdo con esto, ***a(t)*** es una función **creciente** en el tiempo. Todas estas afirmaciones son, naturalmente, hipotéticas y difíciles de fundamentar.

Aceptado todo esto, la hipótesis más simple es que esta función de eficiencia **sea proporcional al tiempo transcurrido**. Nuevamente aquí se introduce otra hipótesis muy difícil de justificar. Pero luego de aceptada esta expresión, se obtiene la ecuación diferencial:

$$\frac{d\,E(t)}{dt} = l\,t.[K - E(t)]$$

Separando variables e integrando se tiene la ecuación:

$$E(t) = K - c\,e^{-\frac{l^2 t}{2}}$$

Esta ecuación cumple que el esfuerzo total es ***K*** para tiempo infinito. Para determinar la constante de integración ***c*** se pude observar, como hipótesis adicional, que en el instante inicial el proyecto puede suponerse que no tiene personal y que la función vale 0. Esta nueva hipótesis tampoco es sencilla de fundamentar. En estas condiciones, queda la ecuación de Rayleigh–Norden, descubierta por Rayleigh en el siglo XIX y revitalizada por Norden para el análisis de proyectos generales [NOR60]:

$$E(t) = K\,[1 - e^{-\frac{l^2 t}{2}}]$$

En definitiva, no puede decirse que este resultado surja de un conjunto de hipótesis plausibles. Es necesario introducir, una y otra vez, hipótesis difíciles de fundamentar.

La utilidad de la ecuación es debido a que sus aplicaciones prácticas son excelentes y su uso es muy simple.

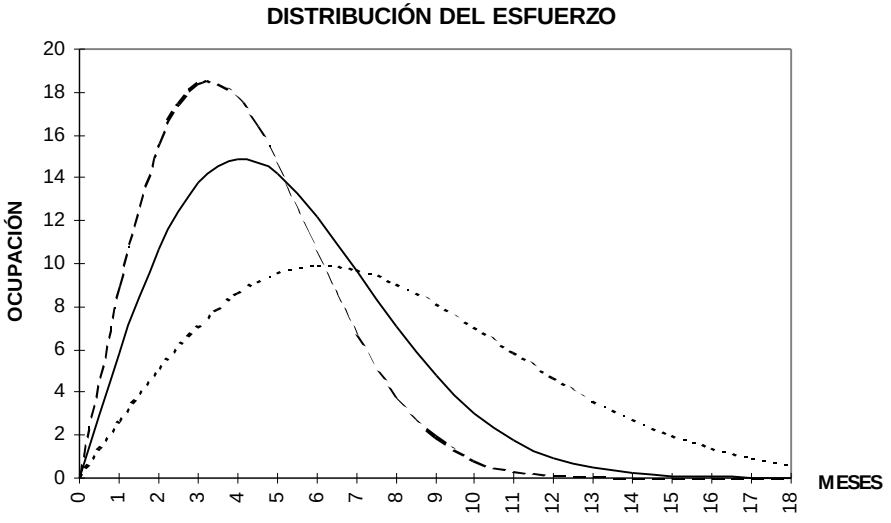
14.4 El modelo de Putnam

La ecuación aceptada hoy para la evolución del esfuerzo de un proyecto es la ecuación de Rayleigh–Norden:

$$E(t) = K\,[1 - e^{-\frac{l^2 t}{2}}]$$

Esta ecuación incluye el esfuerzo total del proyecto, ***K***; pero incluye, además, un parámetro l. Según el valor de este parámetro se tienen diferentes curvas que corresponden a diferentes plazos de entrega. Con esta ecuación, Putnam introduce, en 1978, el análisis dinámico de los proyectos de software.

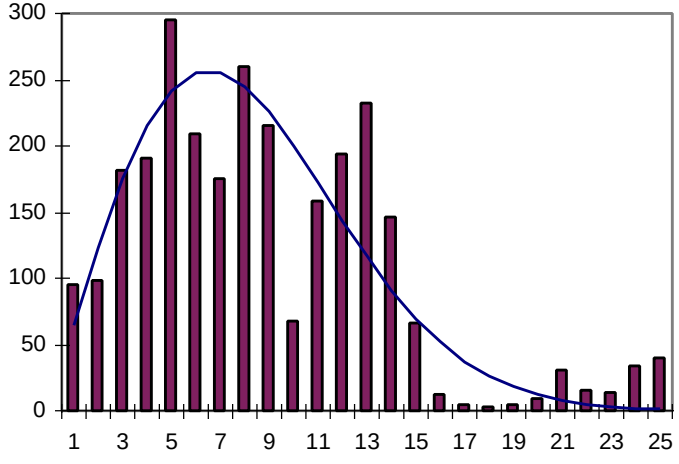
Es interesante presentar un conjunto de curvas para un mismo esfuerzo total de proyecto (100 meses–persona):



Como puede apreciarse, en la curva intermedia –que podemos llamar de velocidad “normal”– se alcanza un máximo de 14 personas ocupadas en el cuarto mes. El plazo de entrega se alcanza en 10 meses. Esta situación es diferente si el proyecto se lo acelera o se lo frena. En la versión “acelerada”, 8 meses de plazo de entrega, para el mismo esfuerzo total se necesitan 18 personas ocupadas en el tercer mes. En la versión “lenta”, 15 meses de plazo de entrega, hay solamente 9 personas ocupadas y este máximo se alcanza en el séptimo mes.

Este tipo de resultados hacen valiosa la ecuación porque permite realizar un estudio simple y rápido de una situación compleja. No cabe duda que la cifra que puede obtenerse de un modelo estático es solamente el total del esfuerzo, 100 meses–persona, pero todavía existen parámetros a ajustar según la manera como se desee llevar el proyecto en el tiempo. Dicho de otra manera, el modelo de Putnam permite ajustar todavía el **plazo de entrega** del proyecto.

Si aplicamos la curva a las horas de trabajo reales del ejemplo de comienzos de este capítulo tendremos la figura:



Se puede apreciar ahora lo que ha sucedido con el proyecto. En el quinto mes se trabajó más de lo previsible y esto fue automáticamente compensado por una disminución del trabajo en los dos meses siguientes. El resto del proyecto marchó bien excepto en el mes 10 en el cual hay una notoria baja dedicación (se trataba de un período de licencias!). Pero esta baja dedicación debió ser compensada por un esfuerzo adicional durante los meses 12, 13 y 14. Finalmente, en el mes 16 se entregó el proyecto (por razones de contrato!) pero no estaba finalizado verdaderamente. Lo ocurrido durante los meses 21 a 25 no es otra cosa que realizar el esfuerzo residual que tenía el proyecto en el momento de entrega y que no se hizo en su momento debido.

Este ejemplo nos muestra no solamente el ajuste de la curva teórica a la realidad sino también la capacidad de análisis que nos da esta herramienta para estudiar la marcha efectiva de un proyecto en ejecución. Es este ajuste con la realidad lo que hace que las curvas de Putnam sean importantes para la gestión de los proyectos.

14.5 Proyecto dividido en módulos

Un importante caso de aplicación del análisis dinámico de Putnam es en el caso de un proyecto dividido en diversos módulos que se realizarán en forma independiente. En estas condiciones, cada uno de los módulos debe ser estudiado como un proyecto independiente. Esto da origen a una curva de Putnam por cada proyecto. De estas curvas parciales puede obtenerse la visión global del proyecto en su totalidad.

Este tema es muy importante, pero no interesa realizar un estudio teórico. En los casos de aplicación se verán algunas situaciones interesantes.

14.6 Uso de la planilla GPS

La planilla GPS posee una hoja dedicada al modelo de Putnam a los efectos de realizar análisis dinámicos de proyectos.

La planilla suministra la función de esfuerzo unitario:

PUTNAM(t; TiempoEntrega)

En esta función ***t*** es la variable de tiempos y ***TiempoEntrega*** es, en las mismas unidades, el plazo de entrega del proyecto.

A los efectos de utilizar esta función es necesario armar una escala de tiempos para calcular, punto a punto, el valor de la curva. Puesto que el valor devuelto corresponde a un esfuerzo unitario, es necesario multiplicar por el esfuerzo total del proyecto, medido en la unidades coherentes de tiempo. Los valores de la curva dan la dedicación en las unidades coherentes que resulten.

A los efectos de considerar algunas situaciones típicas, se tienen los siguientes ejemplos ilustrativos:

- El esfuerzo se mide en **meses–persona**, la unidad de tiempo es el **mes**: en este caso la función calcula cantidad de personas necesarias para cada mes de proyecto.
- Lo mismo sucede si se emplean como unidades de medida **semanas–persona** y **semanas**: la función calcula cantidad de personas.
- El esfuerzo se mide en **horas–persona**, la unidad de tiempo es el **mes**: en este caso la función calcula la cantidad de horas de trabajo que son necesarias realizar en cada mes de proyecto.

La función suministrada por la planilla devuelve el valor cero para tiempos negativos. Esta propiedad es muy importante en el caso de considerar más de un proyecto, con un defasaje en el tiempo. No hay ningún inconveniente en emplear como valor del tiempo, por ejemplo, el valor de una celda **menos** un retardo de comienzo. En este caso, la función será cero hasta el momento en que el proyecto comienza verdaderamente.

La aplicación de la planilla se ejemplifica con casos de estudio.

14.7 Estudio de proyectos en el tiempo

Caso 26: Realización en el tiempo de un proyecto mediano

Tomemos el Caso 25 y analicemos su realización efectiva en el tiempo mediante el modelo de Putnam. La planilla GPS indica, para los valores:

Esfuerzo de desarrollo (meses–persona)	32,3
Tiempo nominal de desarrollo (meses)	9,4
Cantidad promedio de personas	3,4

Se necesitan 5 personas durante el cuarto mes de proyecto. Esta cifra contrasta bien con las 3,4 personas de promedio.

Un error frecuente de los planificadores consiste en confundir **promedio** de personas con **pico** de personas. Supongamos que se decide implementar este

proyecto con 3,5 personas (por ejemplo, con 3 personas en dedicación completa y una a medio tiempo), el modelo de Putnam se debe ajustar, ahora, de modo de no superar esta condición pico. Mediante algo de tanteo en el plazo de entrega se llega a **14 meses de plazo de entrega**. **Esta confusión es la razón más común de no cumplimiento de los plazos de entrega en los proyectos de software**.

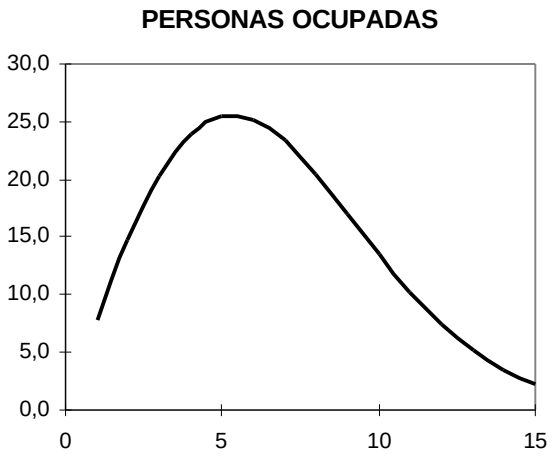
Caso 27: Proyecto grande limitado por el equipo humano

Retomamos el ejemplo analizado en el Caso 9. El análisis estático de COCOMO indicaba:

Miles de líneas de programa	732,5
	Organic
Esfuerzo sin ajustar (meses–persona)	2444,9
Factor de ajuste	1,1
Esfuerzo ajustado (meses–persona)	2689,4
Tiempo de desarrollo (meses)	50,3
Personas en promedio	53,5

Es posible ahora estudiar el desarrollo del proyecto sometido a la limitación de no sobrepasar nunca la cantidad de 25 personas que se disponía para implementarlo. El análisis de Putnam permite ensayar diferentes plazos de entrega de modo de no superar este límite. Luego de algunas pruebas –para lo cual es necesario extender el rango valores de la planilla o cambiar la escala de tiempo a años– se llega un valor aproximado a **160 meses o 13 años**.

El desarrollo en el tiempo previsto por Putnam es:



Es claro que estas no son condiciones reales para un proyecto informático. Nuevamente parece claro que se trata, en realidad, de varios proyectos estudiados como uno único.

Caso 28: Proyectos sucesivos en el tiempo

Consideremos nuevamente el Caso 9, pero ahora como tres proyectos diferentes que se deben realizar en forma sucesiva. El análisis preliminar nos muestra el siguiente cuadro de esfuerzos, tiempos nominales y personas promedio.

	Sistema 1	Sistema 2	Sistema 3
Esfuerzo ajustado (meses–persona)	660,9	1034,6	852,2
Tiempo de desarrollo (meses)	29,5	35,0	32,5
Personas en promedio	22,4	29,6	26,2

Supongamos que la condición es no superar las 25 personas disponibles en el equipo. Es claro que los sistemas 2 y 3 ya lo superan, hasta en promedio. El análisis de Putnam nos indica que también el sistema 1 supera el límite en forma dinámica. Se trata de planificar la acción de modo de poder implementar los sistemas. Supondremos, además, que deber ser realizados en el orden de numeración.

Para esto se arma una nueva planilla que contiene tres columnas de proyecto, los tres esfuerzos, un retardo de comienzo y el tiempo de desarrollo. La planilla posee el aspecto:

		Sistema 1	Sistema 2	Sistema 3	
esfuerzo del sistema		660,9	1034,6	852,2	
retardo de comienzo		0	4	8	
tiempo de entrega		29,5	35	32,5	
	tiempo				TOTAL
	0	0,0	0,0	0,0	0,0
	2	9,0	0,0	0,0	9,0
	4	17,2	0,0	0,0	17,2
	6	24,1	10,0	0,0	34,2
	8	29,2	19,5	0,0	48,7
	10	32,3	27,8	9,6	69,7
	12	33,3	34,7	18,5	86,4
	14	32,5	39,7	26,2	98,3
	16	30,2	42,7	32,3	105,2
	18	26,8	43,9	36,4	107,2
	20	23,0	43,3	38,6	104,9
	22	18,9	41,3	38,8	99,0

Esta planilla es semejante a la hoja de Putnam de GPS, excepto que hay tres columnas, una para cada proyecto. Se han tomado los meses de 2 en 2 dado lo largo del proyecto. Estas columnas son casi iguales a la columna de GPS excepto por una segunda diferencia: la aparición de un **tiempo de retardo**, para tener en cuenta el momento en que el proyecto comienza a implementarse. Esto se tiene en cuenta tomando como argumento de la variable tiempo en la función PUTNAM como la diferencia entre la columna tiempo y el valor constante de la celda que contiene el tiempo de retardo. El resultado, como puede apreciarse con los valores presentados, es que el Proyecto 2 no comienza hasta el mes 4 y el Proyecto 3 no comienza hasta el mes 8.

También se ha agregado una columna de totales de los tres proyectos que será usada en lo que sigue. Con esta disposición podemos comenzar a ajustar los valores de proyecto de acuerdo a lo que nos proponemos hacer. El comienzo del ajuste consiste en ajustar el tiempo de desarrollo del Proyecto 1 de modo de no superar las 25 personas en el pico. Para esto es

cómodo colocar retardos de comienzo grandes en los otros dos proyectos de modo de no interferir: por ejemplo 40 meses en cada uno.

El Proyecto 1, con un tiempo de entrega de 29,5 meses alcanza un pico de más de 33 personas. Si se ajusta este tiempo a 39 meses, se tiene un pico de 25,2 personas que es aceptable.

Corresponde ahora ajustar el Proyecto 2. Por un lado es necesario alargar su tiempo de ejecución para no superar el pico, por otro lado es necesario determinar el momento de comienzo. Llevando el tiempo de entrega a 61 meses se logra que el Proyecto 2 alcance un pico de 25,2 personas, aceptable.

Para ajustar el tiempo de comienzo del Proyecto 2 conviene desplazar el comienzo del Proyecto 3 unos 60 meses. Ahora debemos ajustar el tiempo de comienzo del Proyecto 2 por lo menos 16 meses a los efectos de superar el pico del Proyecto 1. En la columna de TOTAL se observa el personal total empleado en los dos proyectos. Se debe aumentar el tiempo de comienzo del Proyecto 2 hasta el punto en que el total de personas no supere 25, tal como nos proponemos.

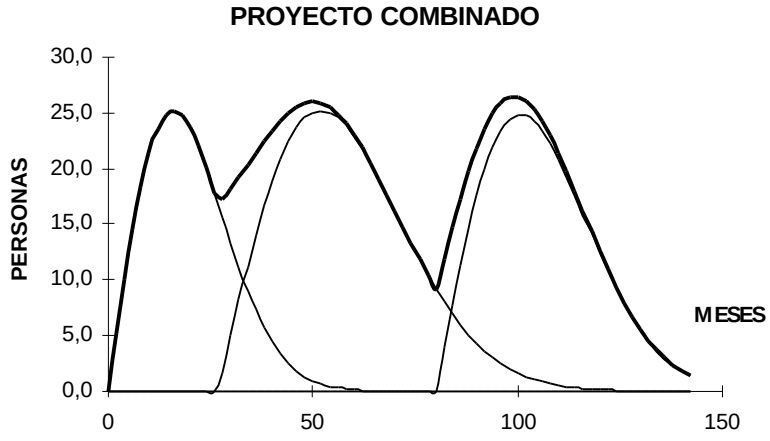
A los efectos del ajuste es cómodo armar una gráfica de cada uno de los proyectos y de la columna de totales. De esta manera es más visible el efecto de ajuste de los parámetros.

Al retrasar el Proyecto 27 meses se logra que el total de personal ocupado alcance un pico de 26 personas, que no está mal para esta etapa del ajuste. Corresponde ahora hacer el mismo proceso con el Proyecto 3. Con un tiempo de entrega de 51 meses no se supera las 25 personas y luego con un retraso de 80 meses se logra no superar la cantidad de personas que se busca. Los valores ajustados corresponden a:

	Sistema 1	Sistema 2	Sistema 3
esfuerzo del sistema	660,9	1034,6	852,2
retardo de comienzo	0	27	80
tiempo de entrega	39	61	51

Entrega del Total 131

La curva del proyecto combinado correspondiente tiene el siguiente aspecto:



El plazo de entrega del proyecto será la suma del retardo del último proyecto, más su plazo de entrega, es decir, 80+51=131 meses. Se ha agregado esta celda en la planilla. Los proyectos analizados de esta manera tienen un plazo de entrega menor que si se los toma en bloque, como se lo hace en el Caso 27.

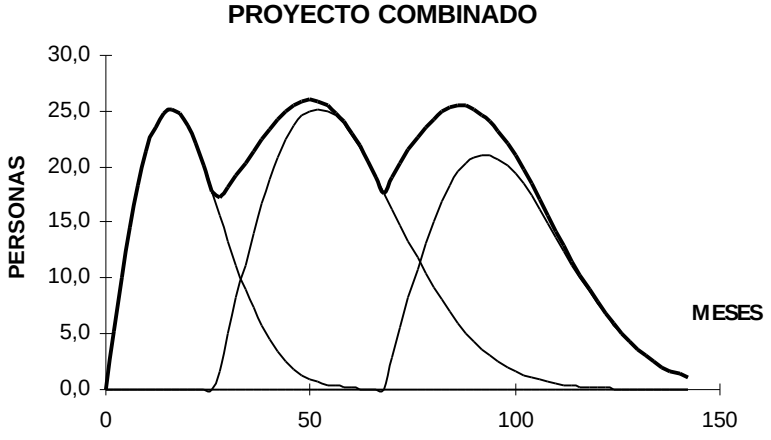
Tal como se puede observar en la curva, el aprovechamiento del equipo humano no es demasiado bueno. En la transición entre el Proyecto 1 y el Proyecto 2 se desciende a 17 personas ocupadas. La situación es peor todavía en la transición entre el Proyecto 2 y el Proyecto 3, donde se desciende a 9 personas ocupadas. Esta situación de debe corregir un poco para tener una ocupación más regular del equipo de trabajo.

La responsabilidad de las caídas de ocupación se debe a que los proyectos se intentan realizar demasiado rápido. Es más conveniente, por ejemplo, **prolongar** el tiempo de desarrollo del Proyecto 3 y **disminuir** su retardo de comienzo, a efectos de mejorar la caída de ocupación detectada. Para esto se procede nuevamente por tanteo.

Con este juego de valores:

	Sistema 1	Sistema 2	Sistema 3
esfuerzo del sistema	660,9	1034,6	852,2
retardo de comienzo	0	27	68
tiempo de entrega	39	61	60
Entrega del Total			128

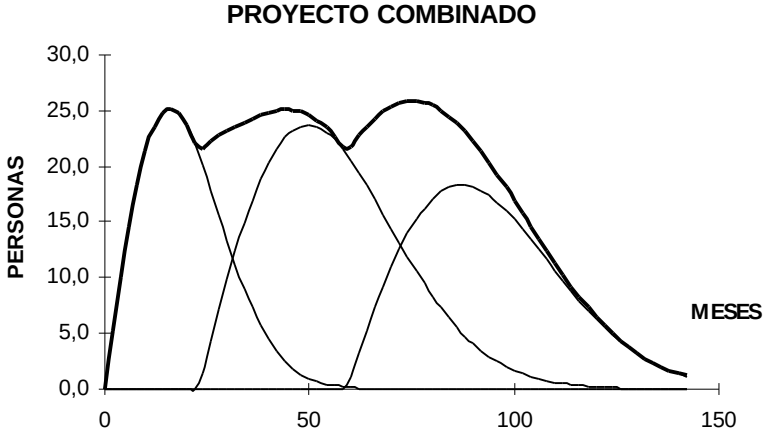
ha descendido el tiempo de entrega del total a 128 meses porque hay un aprovechamiento más parejo de los recursos humanos, tal como muestra la gráfica:



Llegados a este momento, los dos valles se han igualado, ahora es necesario proceder a ajustar también la ejecución del Proyecto 2 por tanteo.
Finalmente, con los valores:

	Sistema 1	Sistema 2	Sistema 3
esfuerzo del sistema	660,9	1034,6	852,2
retardo de comienzo	0	23	59
tiempo de entrega	39	65	69
Total de los proyectos			128

se logra el mismo plazo de entrega (lo cual evidencia que ya no hay muchas posibilidades más de ajuste), pero con una curva de ocupación más pareja:



Con este análisis de Putnam se está en condiciones de realizar la estimación final del proyecto o comenzar a ajustar los detalles efectivos de su implementación mediante el diagrama de Pert y Gantt.

15. CONCLUSIONES

Las metodologías consisten en aplicar **herramientas**, no se les puede pedir milagros. Tampoco se les puede pedir que trabajen solas.

Nada puede reemplazar la información directa recogida de la experiencia pasada. Muchas de las herramientas que se han presentado aquí requieren calibración o criterios para ser aplicadas. Un gerente de proyecto, un analista que realiza un estudio preliminar, todo técnico involucrado en la gestión de proyectos de software debe ser, ante todo, un gran recopilador de información.

No es simple sensibilizar al equipo de trabajo para que haga registros buenos, confiables y completos de los proyectos. Por esta razón, la tarea de recopilar datos es dificultosa. Se debe insistir en todo momento que la recopilación de información no es ni una tarea policial ni una medida del rendimiento individual o colectivo. De todas maneras, aunque los datos recogidos no sean excelentes, es mejor datos imprecisos que ausencia de datos.

No es simple sensibilizar a los jerarcas que vale la pena invertir en estudios de tipo global. Sin embargo los estudios cuestan muy poco y ahorran mucho dinero en el futuro.

No es posible obligar a un jefe de proyecto a que emplee metodologías cuantitativas, modelos de estimación y ecuaciones empíricas. Si no está convencido, las empleará mal y las usará para demostrar que no tienen sentido.

De todas maneras, cualquiera sea el papel que le toque jugar dentro de un proyecto informático, el mejor deseo, luego de todo un libro sobre gestión de proyectos, es: **buena suerte!**

16. MANUAL TÉCNICO DE LA PLANILLA GPS

16.1 Introducción

GPS es una planilla electrónica EXCEL destinada a suministrar las funciones necesarias para la gestión de proyectos de software. Estas funciones están implementadas en Visual Basic.

La planilla posee cuatro hojas, que corresponden a las cuatro técnicas básicas, que se identifican como:

- COCOMO
- PF
- GeneXus
- PUTNAM

La planilla GPS se encuentra protegida por dos mecanismos. Por un lado, no es posible modificar su estructura, de modo que las hojas donde están las funciones no son accesibles. En segundo lugar, la planilla está protegida de la copia mediante una llave de software, tal como se detalla más adelante.

Ninguna de estas protecciones son un obstáculo para el usuario legal de la planilla.

16.2 Funciones que suministra

La planilla GPS suministra un conjunto de funciones que pueden ser encontradas en el grupo de funciones de usuario. Cada función posee una ayuda y sus variables están elegidas de modo que recuerden su significado.

La lista de las funciones es la siguiente:

Kemb (KLOC)
Esfuerzo nominal COCOMO en un proyecto tipo Embedded.

Korg (KLOC)
Esfuerzo nominal COCOMO en un proyecto de tipo Organic.

Ksd (KLOC)
Esfuerzo nominal COCOMO en un proyecto de tipo Semidetached.

Temb (MesesPersona)
Tiempo nominal COCOMO de desarrollo en un proyecto tipo Embedded.

Torg (MesesPersona)
Tiempo nominal COCOMO de desarrollo en un proyecto de tipo Organic.

Tsd (MesesPersona)
Tiempo nominal COCOMO de desarrollo en un proyecto de tipo Semidetached.

xACAP (i) Factor COCOMO. Capacidad de los analistas.
xAEXP (i) Factor COCOMO. Experiencia en la aplicación.
xCPLX (i) Factor COCOMO. Complejidad del producto.
xDATA (i) Factor COCOMO. Tamaño de la base de datos.
xLEXP (i) Factor COCOMO. Experiencia en el lenguaje de programación.
xMODP (i) Factor COCOMO. Empleo de técnicas modernas de programación.
xPCAP (i) Factor COCOMO. Capacidad de los programadores.
xRELY (i) Factor COCOMO. Exigencias de confiabilidad.
xSCED (i) Factor COCOMO. Exigencias de plazo de entrega.
xSTOR (i) Factor COCOMO. Exigencias de memoria.
xTIME (i) Factor COCOMO. Exigencias de tiempo de ejecución.
xTOOL (i) Factor COCOMO. Empleo de herramientas.
xTURN (i) Factor COCOMO. Capacidad de respuesta del ambiente.

xVEXP (i) Factor COCOMO. Experiencia en la máquina lógica.
xVIRT (i) Factor COCOMO. Obsolescencia de la plataforma.

ILF (simple; promedio; complejo)
Internal Logical File, Puntos Funcionales aportados por un archivo lógico interno.

EIF (simple; promedio; complejo)
External Interface File, Puntos Funcionales aportados por una interfaz externa.

EI (simple; promedio; complejo)
External Input, Puntos Funcionales aportados por una entrada externa.

EO (simple; promedio; complejo)
External Output, Puntos Funcionales aportados por una salida externa.

COBOL (PuntosFuncionales)
Esfuerzo nominal en horas para programación COBOL.

GPM (PuntosFuncionales)
Esfuerzo nominal en horas para programación GeneXus.

TDEV (MesesPersona)
Tiempo nominal de desarrollo, en meses, para un sistema de información que requiere un cierto esfuerzo en meses–persona.

PUTNAM (t; TiempoEntrega)
Cantidad de personas, según el modelo de Putnam, empleadas en el instante **t**, para obtener **TiempoEntrega**.

16.3 Protección de la planilla

La planilla GPS está protegida contra copias. Toda vez que se calcule una función, se verificará que la planilla es legítima, si no lo es, el valor devuelto por las funciones es **cero**.

Para realizar este control, la planilla consulta un archivo **KEY.EXE** que debe estar en el directorio donde se encuentra la planilla. No hay ningún inconveniente que este directorio esté en disco duro o en una red.

Originariamente la autorización de uso se encuentra en el disquete de distribución. No hay ningún impedimento a que sea usado directamente e disquete excepto por razones de velocidad o de desgaste del medio. El usuario tiene la opción de **mover** de este archivo de autorización al directorio de trabajo que desee. En este directorio deberá estar también la planilla.

El procedimiento de mover la autorización consta de los siguientes pasos:

- Copiar todo el contenido del disquete al directorio de trabajo.
- Ejecutar el comando **MOVER.EXE <origen> <destino>**: usualmente este comando tendrá como origen la disquetera **a** y como destino el directorio del disco duro que se uso en el paso anterior.
- Se debe responder **Y** a la pregunta de si se desea mover la autorización.

Luego de estas operaciones la planilla quedará operativa en el directorio elegido y no podrá ejecutarse en otro lugar que no se éste.

16.4 Uso combinado con otras planillas

En muchas oportunidades puede interesar trabajar en una planilla que no sea GPS. Esta es una operativa altamente recomendable. Para proceder así basta con abrir la planilla GPS: desde la planilla de trabajo serán visibles todas las funciones necesarias.

Para guardar la planilla de trabajo, es necesario convertir todos los cálculos a valores numéricos mediante las operaciones clásicas de seleccionar, **Copiar** y **Pegado especial** con la opción **Valores**. De otra forma, al cerrar la planilla GPS indicarán error todos los cálculos hechos.

17. ÍNDICE DE CUADROS

18. ÍNDICE DE CASOS

19. BIBLIOGRAFÍA Y REFERENCIAS

[NOR60] P. V. Norden. **On the Anatomy of Development Projects**. IRE Transaction on Engineering Management, Vol. 7, March, 1960.

[BRO72] Frederick P. Brooks, Jr., **The mythical man–month**, 1972.

[PAR80] F. N. Parr. **An Alternative to the Rayleigh Curve for Software Development Effort**. IEEE Transactions on Software Engineering, Vol. 6, N. 3, May 1980.

[BOE81] Barry W. Boehm, **Software engineering economics**, 1981.

[LIP82] M. Lipow. **Number of Faults per Line of Code**. IEEE transactions on Software Engineering, Vol. 8, N. 4, July 1982.

[WAR83] Roger D. H. Warburton, **Managing and predicting the costs of real–time software**, IEEE Transactions on Software Engineering, Vol. 9, September 1983.

[GRO83] Juan Grompone. **Criterios de factibilidad de proyectos en informática**. Anales del XVI congreso Nacional de Informática, p. 20–23, Sao Paulo, Brasil, octubre 1983.

[ALB83] Allan J. Albrecht; John E. Gaffney, **Software function, source lines of code, and development effort prediction: a software science validation**, IEEE Transactions on Software Engineering, Vol. 9, N. 6, November 1983.

[BEH83] Charles A. Behrens, **Measuring the Productivity of computer systems development activities with function Points**, IEEE Transactions on Software Engineering, Vol. 9, N. 6, November 1983.

[BOE84] Barry W. Boehm, **Software engineering economics**, IEEE Transactions on Software Engineering, vol. 10, N. 1, January 1984.

[GRO84] J. Grompone, J. Jerusalmi, , J. Bonfiglio. **Instrumentación de proyectos de software**. Anales de la X conferencia latinoamericana de informática, p. 439–444, Viña del Mar, Chile, abril 1984.

[GRO84a] Juan Grompone. **Aplicaciones de la informática a la ingeniería en América latina**. Proyecto FLAI–UNESCO, contrato N. 561607, abril 1984.

[KEM87] Chris F. Kemerer, **An empirical validation of software Cost estimation models**, Communications ACM, Vol. 30, N. 5, May 1987.

[LON87] Bernard Londeix, **Cost estimation for software development**, 1987.

[SYM88] Charles R. Symons, **Function point analysis: difficulties and improvements**. IEEE Transactions on Software Engineering, Vol. 14, N. 1, January 1988.

[WEY88] Elaine J. Weyuker. **Evaluating Software Complexity Measures**. IEEE Transactions on Software Engineering, Vol. 14, N. 9, September 1988.

[NOR89] Ronald J. Norman; Jay F. Nunamaker Jr. **CASE Productivity Perceptions on Software Engineering Professionals**. Communications ACM, V. 32, N. 9, September 1989.

[KEM89] Rajiv D. Banker and Chris F. Kemerer, **Scale economies in new software development**, IEEE Transactions on Software Engineering, October 1989.

[GRO90] Juan Grompone. **La gestión de proyectos de programación: análisis de 13 años en el Uruguay**. Revista de la asociación de ingenieros. Montevideo, 1990.

[LOW90] Graham C. Low; D. Ross Jeffrey. **Function Points in the estimation and evaluation of the software process**. IEEE Transactions on Software Engineering, Vol. 16, N. 1, January 1990.

[GEN91] Michiel van Genuchten. **Why is Software Late? An Empirical Study of Reasons for Delay in Software Development**. IEEE Transactions on Software Engineering, Vol. 17, N. 6, June 1991.

[JON91] Capers Jones. **Applied software measurements: assuring Productivity and quality**. 1991.

[GHE91] Carlo Ghezzi; Mehdi Jazayeri; Dino Mandrioli. **Fundamentals of Software Engineering**. 1991.

[IFP92] **Function Points as assets**. IFPUG. 1992.

[KEM92] Chris F. Kemerer; Benjamin S. Porter. **Improving the reliability of function point measurement: an empirical study.** IEEE Transactions on Software Engineering, Vol. 18, N. 11, November 1992.

[YOU93] Edward Yourdon. **Decline and fall of the American programmer**. 1993.

[MÖL93] K. H. Möller; D. J. Paulisch. **Software Metrics**. 1993.

[IEE93] **IEEE Standards Collection. Software Engineering**. 1993.

[SHE93] Martin Shepperd; Darrel Ince. **Derivation and Validation of Software Metrics**. 1993.

[KEM93] Chris F. Kemerer. **Reliability of function point measurement: a field experiment**. Communications ACM, Vol. 36, N. 2, February 1993.

[JEF93] D. R. Jeffrey; G. C. Low; M. Barnes. **A comparison of function point counting techniques**. IEEE Transactions on Software Engineering, Vol. 19, N. 5, May 1993.

[IFP94] **Function point counting practices manual (release 4.0)**, IFPUG, January 1994.

[BAN94] Rajiv D. Banker; Robert J. Kauffman; Charles Wright; Dani Zweig. **Automating output size and reuse metrics in a repository–based computer–aided software engineering (case) environment**. IEEE Transactions on Software Engineering, Vol. 20, n. 3, March 1994.

[GRO94] Juan Grompone. **Manual de GPM**. Montevideo, 1994.

[KIT95] Barbara Kitchenham; Shari Lawrence Pfleeger; Norman Fenton. **Towards a Framework for Software Measurement Validation**. IEEE Transactions on Software Engineering, Vol. 21, N. 12, December 1995.

20. ÍNDICE ANALÍTICO

—A—

—B—

—C—

—D—

—E—

—F—

—G—

—H—

—I—

—J—

—K—

—L—

—M—

—O—

—P—

—R—

—S—

—T—

—U—