Departamento de Lenguajes y Sistemas Informáticos

Facultad de Informática

Universidad Politécnica de Madrid

APORTACIÓN A LAS TÉCNICAS DE EVALUACIÓN DE LA PRODUCTIVIDAD DEL SOFTWARE EN LOS ENTORNOS DE PROGRAMACIÓN

Autor ....: Manuel José Barranco García
(Licenciado en Informática)

Directores: Juan Carlos Granja Álvarez
(Doctor en Informática)

J. Ángel Velázquez Iturbide
(Doctor en Informática)

Año ......: 1995

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid, el día ..... de ..... de 19.....

Presidente D. .....

Vocal D. .....

Vocal D. .....

Vocal D. .....

Secretario D. .....

Realizado el acto de defensa y lectura de la Tesis el día ..... de ..... de 19.....

en .....

Calificación: .....

EL PRESIDENTE LOS VOCALES

EL SECRETARIO

RESUMEN DE LA TESIS

Diversos estudios han puesto de manifiesto que el mantenimiento es la actividad que requiere más recursos en un proyecto software. Por tanto, la facilidad de mantenimiento (mantenibilidad) es un factor muy importante en la productividad de un proyecto. Pueden realizarse diversas actividades adicionales durante las primeras etapas de un proyecto para aumentar la mantenibilidad del software. Sin embargo, estas actividades, como cualquier acción de garantía o control de calidad del software, conllevan un coste adicional. Estas actividades deben realizarse de una forma controlada, para evitar gastos excesivos, por lo que la mantenibilidad debe cuantificarse y asignarse según las necesidades.

La tesis propone varios instrumentos de evaluación y mejora de la productividad de un proyecto software. Estos instrumentos tratan de equilibrar los recursos empleados para aumentar la mantenibilidad del producto con los beneficios producidos como consecuencia durante la etapa de mantenimiento. La evaluación de la productividad se basa en dos estimaciones: (a) tráfico de cambio anual en la etapa de mantenimiento y (b) relación entre las características de mantenibilidad y el esfuerzo de mantenimiento. Ambas estimaciones se basan en información histórica de otros proyectos y en el juicio de expertos.

Las técnicas de mejora de la productividad propuestas también tratan la identificación de la información que debe extraerse durante la producción para actualizar la base de datos históricos, así como los momentos en los que la actualización debe hacerse. La tesis se completa con un estudio de la integración de las técnicas en un entorno de programación, así como la aplicación de las técnicas de evaluación a tres proyectos software concretos.

RESUMEN EN INGLÉS (“SUMMARY”)

Several studies show that the maintenance is the activity which requires more resources in a software project. Therefore, the facility of maintenance (maintainability) is an important factor in the productivity of a project. We can doing some additional activities during the former stages of a project in order to improve the software maintainability. However, these activities, like every action of assurance and control of the software quality, involve an additional cost. These activities must be done in a controlled manner, so that excessive costs can be avoided. Therefore, maintainability must be quantified and assigned according to the necessities.

The thesis propose various instruments of evaluation and improvement of the productivity in a software project. These instruments try to equilibrate the resources used to improve the product maintainability and the benefit that this investment produce during the maintenance. The evaluation of the productivity is based in two estimations: (a) traffic of annual change in the maintenance stage and (b) relation between the maintainability characteristics and maintenance effort. These estimations are based on historic information of previous projects and experts’s judgement.

The techniques of productivity improvement that we propose, identify the information that must be extracted during the software production in order to actualise the historic data base, and the moments in which that actualisation must be doing. The thesis is completed with an study of the integration of the proposed techniques in a software environment, as soon as the application to three real software projects.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

1 INTRODUCCIÓN

Uno de los mayores retos que continuamente debe plantearse todo director de proyectos software, es sin duda, el aumento de la productividad, entendida como una medida del rendimiento de los resultados obtenidos como función del esfuerzo aplicado. Este desafío estriba en detectar cuáles son los factores que determinan la productividad, incidir sobre ellos para conseguir un incremento de la misma y medir los resultados obtenidos comprobando que se hayan alcanzando los niveles deseados.

Muchos estudios se han realizado sobre proyectos reales, obteniendo como resultado clasificaciones y listas de los factores que afectan a la productividad. Casper Jones [JONE86] presenta 20 factores tangibles y 25 intangibles. McCall [MCCA77] clasifica los factores según afecten a la operación, a la revisión o a la transición del producto. Conte, Dunsmore y Shen [CONT86] clasifican los factores en cuatro categorías: factores humanos, factores del proceso, factores del producto y factores del sistema informático.

Se considera que la evaluación de la productividad es un tema de gran actualidad. Dada la enorme profusión de empresas creadoras de software, y ante tanta competitividad, un elemento distintivo es la productividad. Aquél que produzca más rápido y mejor, tendrá una enorme ventaja sobre el resto en el mercado del software.

1.1 POSTURAS FRENTE A LAS TÉCNICAS DE EVALUACIÓN DE LA PRODUCTIVIDAD

A menudo estos estudios de modelos y técnicas de evaluación de la productividad no tienen la suficiente aceptación por parte de las empresas de software. Es común que tales empresas ignoren la existencia de dichas herramientas.

Los motivos pueden ser diversos:

- **Desconocimiento**. Aunque en los últimos años se han desarrollado algunos modelos, los beneficios que reportan no han sido suficientemente demostrados, por lo que su conocimiento no se ha extendido de una forma fluida.

- **Difícil adaptación**. La adaptación al entorno del trabajo puede ser traumática. Las interfaces entre herramientas de desarrollo y de evaluación de la productividad pueden ser complicados.

- **Costo añadido**. El empleo de tales técnicas conlleva un costo adicional, cuyos frutos no se tienen a corto plazo. Ante la incertidumbre de tales beneficios es frecuente que la dirección del proyecto sea reacia al empleo de tales técnicas.

Estas ideas se desprenden de la experiencia del doctorando [GRAN95] así como de las opiniones de diversos autores al respecto:

- T. K. Abdel-Hamid [ABDE93]: “Mientras que un número de modelos de estimación han sido desarrollados en los pasados años, su utilidad ha sido pobremente demostrada.”

- N. Fenton [FENT94]: “Los principios de medida básicos, que están bien establecidos en la disciplina de ingeniería tradicional, son sistemáticamente desatendidos en la práctica común del software.”

- J. Ludewing [LUDE94]: “Muy poca gente quiere realmente software de alta calidad. Ellos aprecian la alta calidad de la misma manera que aprecian el agua de Mayo: Es algo enviado desde el cielo, y libremente.”

- J. M. Roche [ROCH94]: “Los intereses de este artículo son la sobreabundancia de métricas del software que han sido propuestas pero que raramente se utilizan en las organizaciones de software ...”

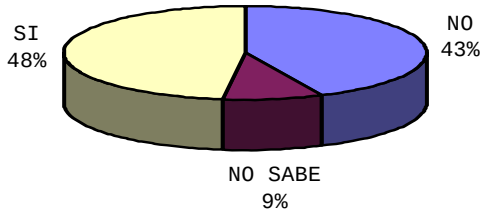
- M. Shepperd [SHEP94]: “Este artículo revela un constante esquema de modelos pobremente articulados y concebidos para tratar las métricas del software. Esto conduce a resultados que son anómalos y fuera de lugar con respecto a la práctica de la ingeniería del software actual”.

Una encuesta recientemente realizada a la totalidad de la plantilla de un departamento de computación corrobora esta idea con los siguientes resultados [HALL94]:

Total encuestados	34
Cuestionarios completos recibidos	23
Porcentaje de respuesta	68%

(A) Actitud hacia las métricas del software.

¿Piensa que la obtención de métricas en Desarrollo de Software es de utilidad?



(B) Proporción de encuestados que opinan que las razones que motivan el rechazo de las métricas del software son las siguientes:

(1) Ausencia de información 92%
(2) Ausencia de entrenamiento 83%
(3) Demasiado trabajo 83%
(4) Ausencia de recursos 79%
(5) Ausencia de procedimientos de trabajo 67%
(6) Ausencia de procedimientos de escritura .. 62%
(7) Procedimientos de gestión pobres 62%

Por las razones expuestas se observa que en el mundo real no está nada claro cómo debe ser un modelo de evaluación de la productividad, ni qué beneficios puede reportar al usuario. A veces, no se sabe qué se desea medir, ni qué unidad de medida emplear. A lo largo de la historia de la ingeniería del software, han sido propuestas varias unidades de medida: tiempo invertido en el desarrollo, coste final del producto, calidad final, etc. Pero, ¿cuál de ellas elegir?. ¿Conviene elegir una sóla o resultará más interesante una mezcla de varias? ¿Qué hacer con las métricas obtenidas? Estos son algunos de los interrogantes que motivan el presente estudio.

Así, para que una técnica de evaluación de la productividad sea debidamente aceptada por una empresa creadora de software, será preciso que quede demostrado:

- **Su eficacia**: La técnica debe cumplir con su objetivo de estimar y medir la productividad, ofreciendo al usuario los medios necesarios para poder incidir sobre ella.

- **Su eficiencia**: Debe ser una técnica *realista*, es decir, su aplicación debe ser rentable para la empresa de software, ya que de otro modo no sería utilizada.

- **Su adaptabilidad al entorno de programación**: La aceptación de una técnica, por parte de un equipo de

software, dependerá de su facilidad de uso. Unas interfaces de usuario claras y potentes revertirán en una mejor acogida de la herramienta por parte del usuario.

Será, pues, preciso examinar la situación actual en la línea recién expresada, y realizar las aportaciones que fueren necesarias para que tal técnica de productividad quede bien definida y demostrada.

1.2 PRODUCTIVIDAD Y MANTENIBILIDAD

En este apartado se va a centrar la problemática que nos ocupa, que el título del apartado resume con dos palabras. A grandes rasgos, se trata de estudiar la relación que existe entre la productividad en un proyecto software y las características de calidad del producto. Más concretamente, el motivo central de esta tesis consiste en análizar el vínculo que sin duda existe entre la productividad en la etapa de mantenimiento y la mantenibilidad (o facilidad de mantenimiento) del producto.

Para introducir el problema, se plantea, en primer lugar, el problema de la etapa de mantenimiento de un proyecto software, situándola como la etapa que más incide sobre el cómputo de la productividad.

Seguidamente se presenta la estrecha relación existente entre calidad y productividad. Se plantea que, efectivamente, la calidad lleva a mejorar la productividad, pero siempre bajo unos límites. Superado dicho límite, los aumentos de calidad apenas inciden en la productividad. Existe por tanto un nivel óptimo de calidad.

Tras estos párrafos introductorios se pasa a plantear y discutir el problema, que como antes decíamos, se centra en la relación existente entre productividad y mantenibilidad.

1.2.1 LA ETAPA DE MANTENIMIENTO

La etapa de un proyecto software que más recursos consume es, sin duda, la etapa de mantenimiento. En ella se emplea entre el 60% y el 80% del coste total del proyecto. Canning [CANN72] comenzó a vislumbrar la problemática de esta etapa del ciclo de vida, al compararla con un iceberg, donde la parte visible no es más que una pequeña parte de la

realidad. Wiener y Sincovec publican sus experiencias con Modula-2 y Ada en [WIEN84] exponiendo que, de acuerdo con estimaciones realizadas en los Estados Unidos, más del 80% del costo del ciclo de vida de un producto software se produce durante el mantenimiento. Harrison y Cook [HARR90] informan también que el presupuesto de tal etapa puede ser del 70% del total. Como se observa, todos ellos coinciden en que más de la mitad (entre el 60% y el 80%) del costo total del proyecto se invierte en la etapa de mantenimiento.

Pickard y Carter, dos investigadores de las universidades estatales de Austin y Misisipí respectivamente, estudian las medidas de la facilidad de mantenimiento [PICK93]. En un párrafo de este artículo dicen: *"... el mantenimiento cuenta con la mayor parte del costo del sistema, y por tanto, provee un gran potencial para reducir el costo total del sistema. Si se puede mejorar la mantenibilidad de un sistema, los costos del mantenimiento total para el sistema pueden ser reducidos".*

En estas citas se observa la magnitud de la etapa de mantenimiento, lo que justifica un estudio de la productividad centrado en la etapa de mantenimiento.

La actividad realizada durante esta etapa del ciclo de vida puede ser considerada como una secuencia de cambios realizados al sistema. Estos cambios, en su gran mayoría, pertenecen a uno de los tres tipos de mantenimiento introducidos por Swanson [SWAN76] y comúnmente aceptados:

(1) **Mantenimiento correctivo**: Cambios hechos para corregir errores.

(2) **Mantenimiento adaptativo**: Cambios hechos para acomodar el sistema a distintas entradas de datos o entornos operativos.

(3) **Mantenimiento perfectivo**: Cambios hechos en respuesta a requerimientos de los usuarios.

Aun existe un cuarto tipo, de incorporación más reciente, introducido por Henry y Wake en [HENR91]:

(4) **Mantenimiento preventivo**: Cambios realizados para reducir la complejidad de un módulo con la esperanza de que las futuras actividades de mantenimiento correctivo puedan ser reducidas.

Todo cambio realizado al sistema pasa necesariamente por tres fases (señaladas por Peercy en [PEER81]):

(a) **Comprensión**.- Determinación de los módulos afectados por el cambio y obtención del conocimiento necesario de cada módulo para realizar el cambio.

(b) **Modificación.-** Realización de los cambios requeridos en cada uno de los módulos afectados.

(c) **Prueba**.- Probar la corrección de los cambios realizados, es decir, que los resultados concuerdan con los requisitos del cambio.

1.2.2 PRODUCTIVIDAD Y CALIDAD DE SOFTWARE

La producción de software de alta calidad puede considerarse como un objetivo importante de la ingeniería del software. Pero este objetivo debe compaginarse con el aspecto económico (al menos en la gran mayoría). Así lo expresa Jochen Ludewing, del departamento de Ciencias de la Computación de la Universidad de Suttgart, en la 4ª Conferencia Europea sobre la Calidad del Software [LUDE94]: *"El último objetivo de la ingeniería del software es maximizar el beneficio neto, es decir, la diferencia entre beneficios y costos. ... Los gastos para software de alta calidad están justificados por sus efectos en los procesos. La calidad debe incrementarse tanto como su efecto en los procesos sea positivo, pero no más allá"*.

E. Orlandi expone un estudio sobre la economía de la calidad del software en la 3ª Conferencia sobre la Calidad del Software [ORLA92]. La Figura 1 resume la aportación de Orlandi que muestra la relación entre calidad y costes. Para representar dicha relación se trazan dos curvas:

CF : Coste que repercuten los fallos
CP : Coste de prevención de fallos

En estas curvas se observa lo siguiente:

a) El coste que supone la mejora de la calidad puede tener un crecimiento exponencial una vez alcanzado cierto valor. La razón es que siempre van a existir ciertos agentes en contra de la calidad del software, impulsados por aspectos de diversa naturaleza (económicos, tecnológicos ...), cuya presión, como es lógico, será mayor cuanto mayor sea el nivel de calidad que se pretende alcanzar.

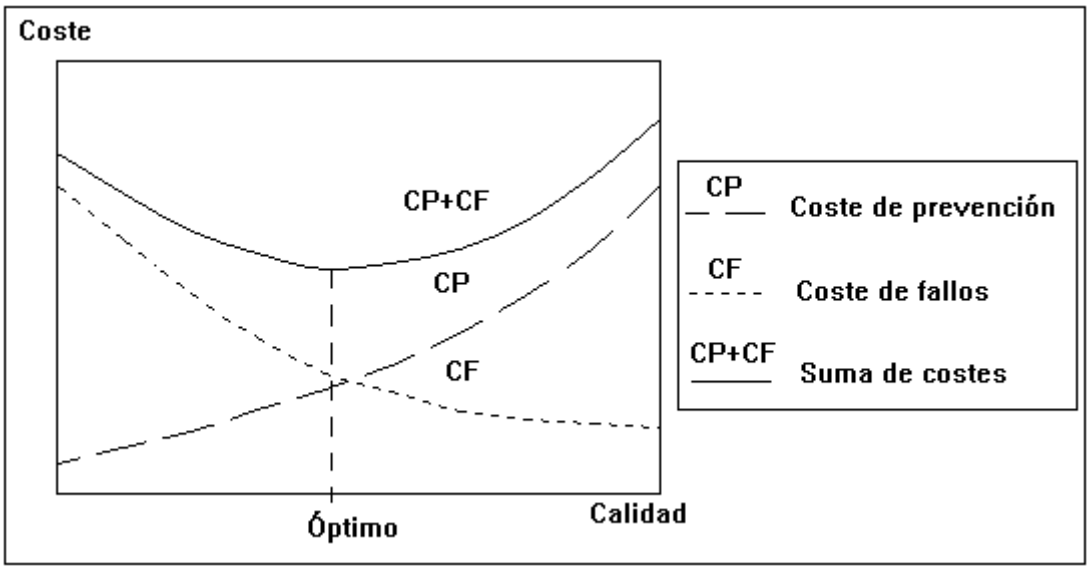


Figura 1.- Coste de la calidad en el software

b) Según aumenta el nivel de calidad, el coste de los fallos va a ir disminuyendo de forma asintótica. Es decir, va a existir un límite asintótico para dicho coste, de forma que a partir de cierto nivel de calidad los incrementos van a ser escasos.

Como se puede ver en el gráfico, la curva CP+CF que recoge la suma de ambas curvas, va a tener un mínimo que corresponderá al mejor valor de calidad, es decir, aquel valor que implica el menor coste global (suma de costes de prevención y repercusiones de fallos).

Resumiendo este apartado diremos que la calidad del software debe estar supeditada al principal objetivo de costes de la ingeniería del software, y por tanto, aquellas acciones de asignación de calidad que penalicen tal objetivo deben ser cuestionadas.

1.2.3 EL PROBLEMA DE LA PRODUCTIVIDAD EN LA ETAPA DE MANTENIMIENTO

Las declaraciones anteriores referentes a la magnitud de la etapa de mantenimiento en los proyectos software, llevan a considerar que esta etapa debe ocupar un lugar preferente en los estudios de productividad. Las cifras expuestas indican que en la medida en que se intervenga sobre la productividad en el mantenimiento se estará

actuando sobre más de la mitad del esfuerzo total invertido en el proyecto. Es por esto que el problema abordado en la presente tesis se centra en el estudio de la productividad en la etapa de mantenimiento.

La **mantenibilidad** o facilidad de mantenimiento es el factor de calidad del software que condiciona la productividad durante dicha etapa. D. Frost lo define [FROS85] como *una medida del costo y dificultad que supone el mantenimiento en un proyecto software.* Existe una relación directa entre dicho factor de calidad y el costo del mantenimiento. Así lo corroboran Martin y McClure [MART83], Banker [BANK93] y Pickard [PICK93].

Hablamos de **características de mantenibilidad** para referirnos a aquellos aspectos del software que no afectan a su funcionalidad sino a su mantenibilidad. Es decir, son aspectos cuya alteración no repercute en un cambio del funcionamiento del software, aunque sí va a tener influencia en las tareas de realización de cambios durante el mantenimiento.

Considerando las tres fases necesarias para realizar un cambio durante el mantenimiento, la mantenibilidad o facilidad de mantenimiento, se puede descomponer en tres elementos, según se expresa en la Figura 2:

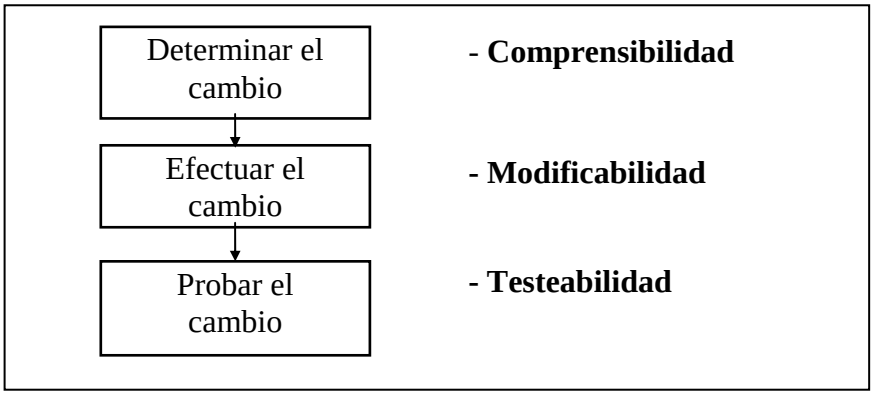


Figura 2.- Componentes de la mantenibilidad

Así, una característica de mantenibilidad corresponderá a uno u otro componente según afecte a una u otra fase de la realización de cambios.

Muchas de las características de mantenibilidad tienen un costo asociado. Es decir, el desarrollo del producto sin tales características resulta más barato que con ellas. Dicho esfuerzo extra, que denominamos **esfuerzo de desarrollo y control** está constituido por tres componentes (véase

Figura 3): (a) esfuerzo inicial de desarrollo, (b) esfuerzo de control y (c) esfuerzo de corrección.

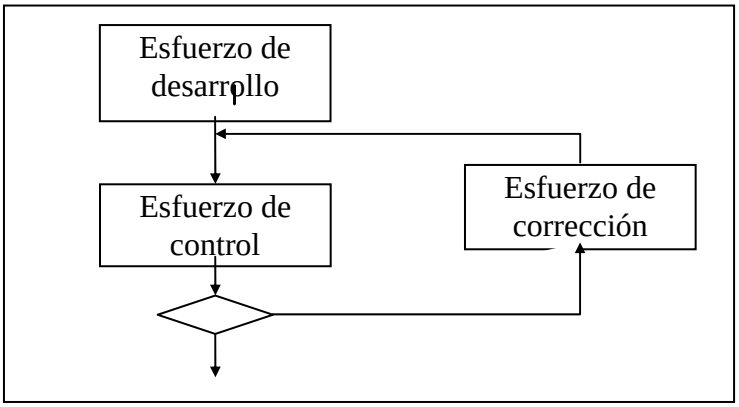


Figura 3.- Aplicación de un elemento de calidad

El esfuerzo de desarrollo y control viene dado por la expresión:

EDC = EDS + ECT + N·(ECR + ECT)

donde,

EDC = Esfuerzo de producción y control
EDS = Esfuerzo de desarrollo inicial
ECT = Esfuerzo de control
ECR = Esfuerzo de corrección
N = Número de iteraciones (0,1,2...)

Considérese, por ejemplo, el costo de incorporar unas líneas de comentario a un programa fuente con el propósito de que su comprensión sea más fácil durante el mantenimiento. Esta actividad conlleva un costo de **desarrollo** evidente. Por otro lado, el aseguramiento de cualquier característica de calidad en un producto software requiere una actividad de **control**, con el fin de comprobar que los requerimientos de calidad han sido alcanzados.

La mantenibilidad es un factor de calidad que responde a este esquema. Por un lado va a suponer un costo extra durante las etapas de desarrollo, y por otro, va a reportar un beneficio que no se tendrá hasta llegar a la etapa de mantenimiento. Aquí surge, de forma inmediata, una cuestión: ¿hasta qué punto deja de ser “conveniente” la asignación de mantenibilidad a un producto software? Según lo expresado en el apartado de aproximación al problema, existe un punto, a partir del cual, la dotación de calidad al producto resulta improductiva.

El gráfico elaborado por Orlandi (vease Figura 1), muestra cómo existe un valor óptimo de asignación de calidad

(en nuestro caso mantenibilidad), considerando los costes de desarrollo y los beneficios en la etapa de mantenimiento. Es decir, cuando la inversión en mantenibilidad no se vea compensada por un beneficio durante el mantenimiento, se tratará de una inversión inútil.

Concretando el problema, **se trata de evaluar la relación que existe entre la mantenibilidad del producto software y la productividad durante la etapa de mantenimiento**. Y como es evidente, el conocimiento de dicha relación va a estar unido a la posibilidad de determinar los niveles de mantenibilidad que ofrecen una mejor productividad en el mantenimiento. Los siguientes planteamientos muestran el problema de forma más precisa:

(a) **Estimación de la productividad en el mantenimiento a partir de una medida de la mantenibilidad**. Se debe disponer de cierto instrumento de evaluación que permita estimar la productividad en el mantenimiento tomando como referencia alguna medida de la mantenibilidad de los componentes software.

(b) **Obtención de la asignación más adecuada de características de mantenibilidad**. Una asignación de características de mantenibilidad a todos los módulos por igual puede resultar poco interesante, ya que no todos los módulos van a verse afectados del mismo modo por los cambios.

1.3 OBJETIVOS DE LA TESIS

En resumen, y considerando lo dicho hasta el momento, los objetivos de esta tesis se plantean en los siguientes términos:

1) Diseño de una técnica o conjunto de técnicas que permitan evaluar y ofrezcan las pautas a seguir para mejorar la productividad en la etapa de mantenimiento:

1.a) Técnica de evaluación de la productividad en el mantenimiento.

1.b) Técnica de obtención de lo valores óptimos de mantenibilidad para cada componente del sistema.

2) Integración de las técnicas desarrolladas en entornos de programación.

2 SITUACIÓN ACTUAL

En este capítulo se realiza una revisión del estado del arte con respecto al problema planteado. En primer lugar se presentan algunos estudios de productividad clásicos y otros más recientes. Después se exponen otros factores que también inciden en la productividad de forma más o menos directa.

2.1 ESTUDIOS DE PRODUCTIVIDAD

Exponemos los estudios más recientes sobre productividad, así como algunos estudios clásicos. En primer lugar se incluyen los estudios sobre los factores que afectan a la productividad del software. Más tarde se presentan los modelos de evaluación de la productividad desarrollados hasta el momento. Tras ello se pasa a exponer los estudios que relacionan la mantenibilidad con la productividad, presentando lo que dicen diversos autores sobre la necesidad de garantía de mantenibilidad. Luego se muestran algunos trabajos sobre métricas de mantenibilidad y finalmente se incluyen algunas declaraciones que revelan el papel fundamental que debe jugar la comprensibilidad en todo estudio de mantenibilidad del software.

2.1.1 FACTORES DE PRODUCTIVIDAD

A lo largo de la historia se han realizado varios estudios acerca de los **factores que inciden en la productividad**. Estos estudios se limitan a relacionar los factores de los que depende la productividad, expresando de qué modo incide cada uno de ellos sobre la misma. Todo ello en base a una información histórica de proyectos anteriores, y en algunos casos (como en el modelo de Boehm) se sirven del juicio de expertos. Seguidamente se presentan los que consideramos más importantes.

- Harr realizó un estudio sobre un conjunto de proyectos de la Bell Telephone (vease [HARR69]), donde se puede apreciar que el **tipo de proyecto** afecta a los valores de la productividad.

Tabla 1.- Resultados del estudio de Harr

Tipo de programa	Nº de módulos	Nº de personas	Personas año: P-A	Tamaño en palabras	Palabras por P-A
Operación/ Control	50	83	101	52,000	515
Mantenimto./Control	26	60	81	51,000	630
Compilador	13	9	17	38,000	2,230
Traductor	15	13	11	25,000	2,270

Se observa una productividad muy baja en los proyectos de control. Esto podría interpretarse de varias maneras: baja productividad en equipos numerosos, o elevada complejidad de este tipo de software.

- En 1983 Wang (vease [WANG84]) realizó un experimento en el cual 44 estudiantes, todos ellos con conocimientos similares, realizaron un desarrollo de software partiendo de las mismas especificaciones. Todos ellos tenían conocimientos similares y disponían de las mismas herramientas, por tanto, las diferencias de productividad que se observaron sólo eran atribuibles a la **capacidad personal** de los participantes.

- Un estudio interesante, considerando el volumen de proyectos examinados, fue el realizado por Walston y Felix (vease [WALS77]). Se consideraron 51 proyectos diferentes escritos en 28 lenguajes distintos. Se identificaron 29 factores con una incidencia importante en la productividad.

Algunos de los factores más importantes obtenidos tras este estudio son:

- Complejidad de las interfaces.
- Participación del usuario en la especificación de requerimientos.
- Experiencia en el área de aplicación.
- Experiencia y cualificación de personal.
- Experiencia en aplicaciones de tamaño y complejidad similar o superior.
- Número de páginas de documentación por cada mil líneas de código.

- El modelo COCOMO es un modelo de estimación de costes propuesto por Boehm en 1981 [BOEH81], obtenido a partir de información de 63 proyectos escritos en 7 lenguajes distintos, que analiza una serie de factores denominados CDA (“Cost Drivers Attributes”) con un

importante impacto sobre la productividad. Se agrupan en cuatro categorías:

Atributos del software
RELY: Fiabilidad del software requerida.
DATA: Tamaño de la base de datos.
CPLX: Complejidad del producto.

Atributos del hardware
TIME: Restricciones de rendimiento en tiempo de ejecución.
STOR: Restricciones de memoria.
VIRT: Volatilidad del entorno de la máquina virtual.
TURN: Tiempo de respuesta requerido.

Atributos del personal
ACAP: Capacidad de análisis.
AEXP: Capacidad del ingeniero de software.
PCAP: Experiencia con las aplicaciones.
VEXP: Experiencia con la máquina virtual.
LEXP: Experiencia con el lenguaje de programación.

Atributos del proyecto
MODP: Utilización de herramientas software.
TOOL: Aplicación de métodos de ingeniería del software.
SCED: Planificación temporal del desarrollo requerida.

Este modelo asigna a cada atributo un rango de productividad y hasta seis valoraciones posibles. La valoración determina un valor numérico, que al ser multiplicado por el rango se obtiene la incidencia de dicho atributo sobre la productividad.

A diferencia del estudio de Walston y Felix, netamente estadístico, el modelo de Boehm se sirve en buena parte del **juicio de expertos**.

2.1.2 MODELOS DE PRODUCTIVIDAD

A diferencia de los modelos de productividad comentados en el subapartado anterior, que muestran factores de incidencia, otros modelos realizan además estimaciones y mediciones de la productividad.

• El modelo SPQR (“Software Productivity, Quality and Reliability”) [JONE86] es un modelo de estimación cuyo objetivo consiste en obtener valoraciones del esfuerzo preciso para desarrollar un proyecto, así como de la productividad y la fiabilidad que presentará.

El modelo precisa de una **base de datos** que debe contener **antecedentes históricos** del desarrollo a realizar. El algoritmo central del modelo utiliza las siguientes variables y cálculos básicos:

TP: Tamaño del producto (en líneas)
EA: Envergadura de asignación, o porción del producto que puede ser asignada a cada miembro del equipo de producción.
IP: Índice de producción (líneas por unidad de tiempo).
TE: Tamaño del equipo

$$TE = \frac{TP}{EA}$$

E: Esfuerzo

$$E = \frac{TP}{IP}$$

T: Tiempo

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

Además de estos datos básicos, se solicita otro conjunto de datos entre los que cabe destacar:

Tipo de desarrollo: programa nuevo, ampliación o mantenimiento.

Principios buscados en el desarrollo: Más calidad, menos coste, menos tiempo de desarrollo, etc.

Tipo de proyecto: por lotes (“batch”), interactivo, científico, gestión, etc.

Novedad del proyecto.

Experiencia del personal.

Grado de reusabilidad.

Complejidad: de los datos y estructural.

Nivel del lenguaje.

Tras introducir todos los datos solicitados, el modelo da como resultado estimaciones de tamaño, costes, esfuerzos, personal, riesgos, errores, calidad y productividad.

- El objetivo del proyecto **SPEM** [SPEM88, SPEM90A, SPEM90B] era obtener una medida a posteriori de la productividad. El modelo recoge información objetiva ——análisis del código y su entorno—— e información subjetiva ——valoraciones de los responsables del desarrollo——, para calcular un índice y unos estimadores de productividad.

La visión de la productividad de este modelo difiere del enfoque tradicional, ya que se obtiene como la relación entre el coste real y el esperado según las condiciones del entorno.

Este modelo se desarrolló en tres fases:

a) Recogida de datos.
b) Análisis estadístico.
c) Elaboración del modelo.

El gran inconveniente que se hubo de afrontar durante el desarrollo de este modelo fue el reducido tamaño de la base de datos (sólo 19 proyectos). Esta es la principal razón por la que los valores estadísticos obtenidos tuvieron que ser sometidos a técnicas de juicio de expertos, con objeto de ajustar la valoración a la realidad.

Dicho inconveniente también tuvo como consecuencia el limitado tamaño del conjunto de CDA's ("Cost Driver Attributes"):

TM: Herramientas y métodos
HC: Complejidad jerárquica
RQ: Calidad de los requerimientos
CP: Capacidad del equipo
MA: Accesibilidad de la máquina
NV: Novedad del proyecto
CT: Restricciones del proyecto

En los análisis de los datos se observó una importante dependencia entre las estimaciones y el tipo de aplicación. Es por ello, que se establecieron tres conjuntos:

Conjunto G: Conjunto global (19 proyectos).
Conjunto B: Aplicaciones de gestíon (9 proyectos).
Conjunto S: Aplicaciones de sistema (7 proyectos).

Los CDA's considerados en cada conjunto de proyectos son los siguientes:

CDAS(G)={RQ,TM,CP,HC}
CDAS(B)={NV,RQ,TM,CP,HC}
CDAS(S)={CT,RQ,TM,CP,MA}

El índice de productividad propuesto,

$$PI = \frac{SE}{AE}$$

relaciona el esfuerzo estimado o estándar (SE o "Standard Effort") y el real (AE o "Actual Effort").

El esfuerzo real se obtiene a partir de una sencilla contabilidad por parte del gestor del proyecto.

Para obtener el esfuerzo estimado, se tomó como base el modelo COCOMO. Así, el cálculo del esfuerzo estimado o estándar viene dado por la expresión:

$$SE = a\ NS^b \cdot c_1 \cdot c_2 \cdot \ldots$$

donde:

a y b son constantes dependientes del conjunto de proyecto,
NS es el número real de instrucciones y
c_i son los CDA's.

Un aspecto importante en este modelo es la obtención de los valores estándares de los CDA's del entorno, a saber, **TM, CP** y **MA** (aunque el último sólo se considera en el conjunto S), ya que con ellos se define el entorno óptimo donde se obtienen los valores de productividad estándar.

Como se puede observar, un elemento fundamental en todo estudio de productividad es la **información histórica**. Sin dicha información, no sería posible detectar los factores determinantes de la productividad.

En algunos modelos también se incorpora otro elemento, que también consideramos fundamental, como es el **juicio de expertos**. Es evidente que siempre habrá notables diferencias entre el proyecto a emprender y los proyectos en los que se basa la información histórica. La intervención de personal debidamente cualificado y experimentado constituye un elemento primordial para la interpretación correcta de los datos históricos.

Todos los modelos coinciden en considerar como unidad de producción, la **línea de código** (o múltiplo). Así, la unidad de productividad empleada viene dada por el cociente entre líneas de código producidas y unidad de tiempo.

En el último modelo expuesto, se considera un **índice de productividad** construido como el cociente entre el esfuerzo estimado y el real, perfecto indicador de la bondad de la estimación.

Considerando la gran importancia de la etapa de mantenimiento en los proyectos software (como se ha expresado en el apartado 1.2.1), vamos ahora a revisar algunos modelos de estimación del esfuerzo en dicha etapa:

- El modelo clásico de evaluación del esfuerzo de mantenimiento propuesto por Belady y Lehman [BELA72] es el siguiente:

$$M = p + Ke^{(c-f)}$$

donde:

M = esfuerzo total de mantenimiento
p = esfuerzo productivo (análisis y evaluación, modificación, codificación)
K = una constante empírica
c = una medida de la complejidad que se puede atribuir a la falta de un buen diseño y de documentación
f = una medida del grado de familiaridad con el software

Esta medida indica que el esfuerzo de mantenimiento puede aumentar exponencialmente cuando se usa un pobre enfoque de desarrollo de software (elevada complejidad del software) o si la familiaridad con el software es escasa.

- El modelo COCOMO propuesto por Boehm [BOEH81] ofrece una estimación del esfuerzo en el mantenimiento basado en el parámetro **Tráfico de Cambio Anual** (TCA), que es la proporción de instrucciones fuente que sufren algún cambio durante un año, bien sea por adición o por modificación.

$$TCA = \frac{NLN + MLM}{NLI}$$

siendo,

NLN = Número de líneas nuevas
NLM = Número de líneas modificadas (incluidas las modificaciones sobre las incorporadas durante el mantenimiento)
NLI = Número de líneas inicial (antes de comenzar la etapa de mantenimiento)

La ecuación para estimar el esfuerzo de mantenimiento anual $(MM)_{MANT}$ una vez conocido el esfuerzo de desarrollo estimado $(MM)_{DES}$ viene dado por:

$$(MM)_{MANT} = TCA\ (MM)_{DES}$$

siendo MM ("Month Man") una unidad de esfuerzo consistente en el trabajo de una persona durante un mes, 19 días o 156 horas.

El modelo COCOMO propone un conjunto de ecuaciones, obtenidas empíricamente, para la estimación del esfuerzo de desarrollo. Al aplicar la versión inicial a una amplia variedad de entornos se comprobó que no bastaba con un único modo de desarrollo, por lo que se plantearon tres modos (orgánico, "semidetached" y "embedded"), de modo que se pudiesen realizar estimaciones con un buen grado de precisión en cualquier entorno de desarrollo.

El modo *orgánico* se considera cuando el equipo de desarrollo es pequeño y trabaja en un entorno familiar, con experiencia en proyectos similares. No existen grandes necesidades de comunicación y se negocian fácilmente las modificaciones que se precisen en cada momento. Suelen ser proyectos de tamaño en torno a los 50 KS.

El modo *"semidetached"* (semidesligado o semilibre) es intermedio entre el orgánico y el "embedded", por lo que es como una mezcla de los dos: hay un nivel intermedio de experiencia en proyectos similares, mezcla de gente experta e inexperta, y mezcla de elementos rígidos y flexibles. Suelen tener tamaños en torno a los 300 KS.

El modo *"embedded"* (restringido) se aplica a proyectos en los que se debe trabajar con fuertes restricciones, con entornos software-hardware fuertemente acoplados. Es muy difícil la negociación de cambios en las especificaciones del software. Generalmente abarcan áreas menos conocidas que en los otros casos.

Para cada modo de desarrollo existen tres versiones del modelo: básica, intermedia y detallada.

El modelo *básico* es adecuado para realizar estimaciones rápidas, aunque sin gran precisión, pues el único parámetro que utiliza es el tamaño del producto.

El modelo *intermedio* considera 15 atributos (ya relacionados en el apdo. 2.1.1) cuya valoración actúa como factor multiplicador en el modelo.

El modelo *detallado* considera las estimaciones de esfuerzo en cada etapa del ciclo de vida del proyecto.

Las ecuaciones de estimación de esfuerzo en el COCOMO *básico* son:

Modo de desarrollo	Esfuerzo
Orgánico	$MM = 2.4\ (KS)^{1.05}$
Semidetached	$MM = 3.0\ (KS)^{1.12}$
Embebedded	$MM = 3.6\ (KS)^{1.20}$

KS = tamaño del programa en miles de líneas de código fuente.

2.1.3 GARANTÍA DE MANTENIBILIDAD

Un factor de calidad a tener muy en cuenta en los estudios de productividad es la **facilidad de mantenimiento** o **mantenibilidad.** Antes de revisar las métricas propuestas por diversos autores, veamos las opiniones referentes a la importancia de la mantenibilidad y cómo garantizarla.

- B. Boehm [BOEH79] realizó un estudio que ponía de manifiesto el enorme costo que supone el mantenimiento de un software difícilmente mantenible. Sus estudios indicaban que puede haber una relación de 40 a 1 entre el esfuerzo de mantenimiento en tales circunstancias y el esfuerzo en nuevos desarrollos.

- H. Schäfer [SCHÄ84] coincide con Boehm, y recomienda las siguientes reglas respecto a qué hacer cuando un módulo es difícilmente mantenible:

  a) Destruirlo y desarrollarlo de nuevo si es pequeño.

  b) Modificar el módulo si es grande, complicado o altamente acoplado con otros módulos.

  Sus estudios muestran que se puede ahorrar hasta un 45% del costo del mantenimiento siguiendo estas reglas.

- Con el objetivo de garantizar la mantenibilidad, Parnas [PARN79] recomendaba lo siguiente:

  a) Estructurar los requerimientos de cambios en componentes, separando aquellos que puedan ser objeto de futura expansión.

  b) Hacer uso de niveles de abstracción cuando se diseña un producto. Cuando se realizan alteraciones, las funciones pueden ser añadidas o eliminadas sin efectos laterales.

La primera recomendación de Parnas puede ponerse en práctica planificando el mantenimiento mediante el empleo del control de versiones (una de las actividades de la gestión de configuraciones software). Existen dos clases de modelos de control de versiones:

- Modelos clásicos u orientados a versión [CLEM89, PLAI93], en los que los cambios provocan la aparición de nuevas versiones del sistema, identificándose cada nueva versión.

- Modelos orientados al cambio [LIE89] en los que una versión del sistema se identifica a través de los cambios que incluye.

La segunda recomendación de Parnas conduce a una estructura multinivel de los sistemas software. Una propuesta en este campo realizada por E. Denert y R. Thurner (expuesta en [WALL94]) consiste en estructurar los sistemas software en cuatro niveles, según se muestra en la Figura 4.

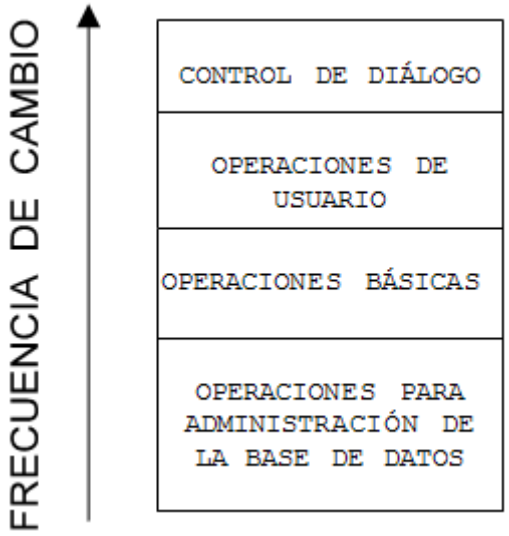


Figura 4.- Estructuración multinivel del software

1.- **Operaciones para administración de la base de datos**. Este nivel reúne todas las operaciones de manejo de los datos, que pueden estar organizados en ficheros o en base de datos: localización, inserción, modificación, eliminación, etc.

2.- **Operaciones básicas**. Operaciones de preparación de datos o que disponen los datos en un orden determinado.

3.- **Operaciones de usuario.** Aquí están incluidas las operaciones propias de la aplicación.

4.- **Control de diálogo**. Este nivel contiene el monitor de diálogo con el usuario. El usuario controla el diálogo usando menús y comandos.

Como se observa en la Figura 4, la frecuencia de cambios va creciendo según se asciende de nivel, lo cual es bastante evidente. Considerese que los dos primeros niveles serán comunes a distintas aplicaciones y serán, por tanto, bastante estables. El tercer nivel contendrá la descripción de los datos de cada aplicación y las operaciones que sea preciso programar. Será por tanto un nivel que podrá cambiar con cierta frecuencia, pero siempre menos que el último nivel que es el que se encuentra más próximo al usuario, controlando formas de entrada de datos, formas de presentación de datos, etc.

2.1.4 MÉTRICAS DE MANTENIBILIDAD

Seguidamente se expresan las opiniones de varios autores referentes a métricas de mantenibilidad:

- Un estudio realizado por W. Itzfeld en Alemania y recogido por Wallmüller en [WALL94], realiza el siguiente "ranking" de utilización de métricas de calidad:

Tabla 2.- Ranking de utilización de las métricas de calidad

%%	Métrica de calidad
67%	Mantenibilidad
65%	Facilidad de uso
60%	Fiabilidad
40%	Corrección
19%	Eficiencia
15%	Portabilidad
17%	Otras

%% Proporción de encuestados que emplean alguna métrica del tipo indicado.

Como se observa en la Tabla 2, las métricas de mantenibilidad ocupan la primera posición en este "ranking", lo cual pone de manifiesto la gran importancia de este factor de calidad.

• Kafura y Reddy [KAFU87] examinan un amplio conjunto de métricas en conexión con la etapa de mantenimiento, llegando a la conclusión de que las métricas constituyen un excelente instrumento para:

a) Observar la complejidad creciente de un sistema.

b) Identificar los componentes de un sistema que estén mal estructurados.

• Curtis [CURT80], ampliando un estudio previo de métricas de calidad realizado por Boehm, considera el atributo *mantenibilidad* a través de sus tres componentes:

- Facilidad de comprensión (o comprensibilidad)
- Facilidad de modificación (o modificabilidad)
- Facilidad de prueba (o testeabilidad)

Un conjunto de variables o factores medibles determinan cada uno de estos componentes o atributos de la mantenibilidad. Dichas variables se muestran en la Tabla 3.

Consistencia.- Uso de un diseño uniforme y de técnicas de documentación a lo largo del proyecto de desarrollo del software.

Accesibilidad.- Grado en que todos los recursos del sistema son accesibles.

Tabla 3.- Relación entre atributos y métricas según Boehm-Curtis

Atributo: / *Métrica:*	*Comprensibilidad*	*Modificabilidad*	*Testeabilidad*
Consistencia	X		
Accesibilidad			X
Estructuración	X	X	X
Autodescripción	X		X
Concisión	X		
Legibilidad	X		
Expandibilidad		X	

Estructuración.- Empleo de técnicas de diseño y programación estructurada.

Autodescripción.- Grado en que el programa muestra su propio funcionamiento así como los errores que aparecen.

Concisión.- Lo compacto que es el programa en términos de líneas de código.

Legibilidad.- Grado de claridad que posee el programa en cuanto a nombres de variables, aspectos de estilo, etc.

Expandibilidad.- Grado en que se puede ampliar el diseño arquitectónico, de datos o procedimental.

• McCall propone el siguiente conjunto de métricas de mantenibilidad [MCCA77]:

Concisión.- Lo compacto que es el programa en términos de líneas de código.

Consistencia.- Uso de un diseño uniforme y de técnicas documentación a lo largo del proyecto de desarrollo del software.

Instrumentación.- Grado en que el programa muestra su propio funcionamiento e identifica los errores que aparecen.

Modularidad.- Independencia funcional de los componentes del programa.

Autodocumentación.- Grado en que el código fuente proporciona información significativa.

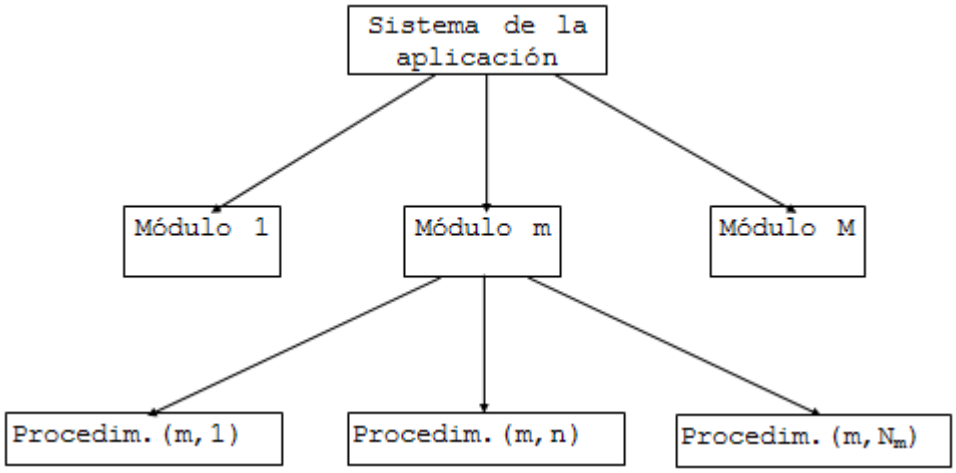
Sencillez.- Grado en que un programa puede ser entendido sin dificultad.

Ambas relaciones de métricas de mantenibilidad tienen bastante en común, y serán tenidas en cuenta en el presente estudio.

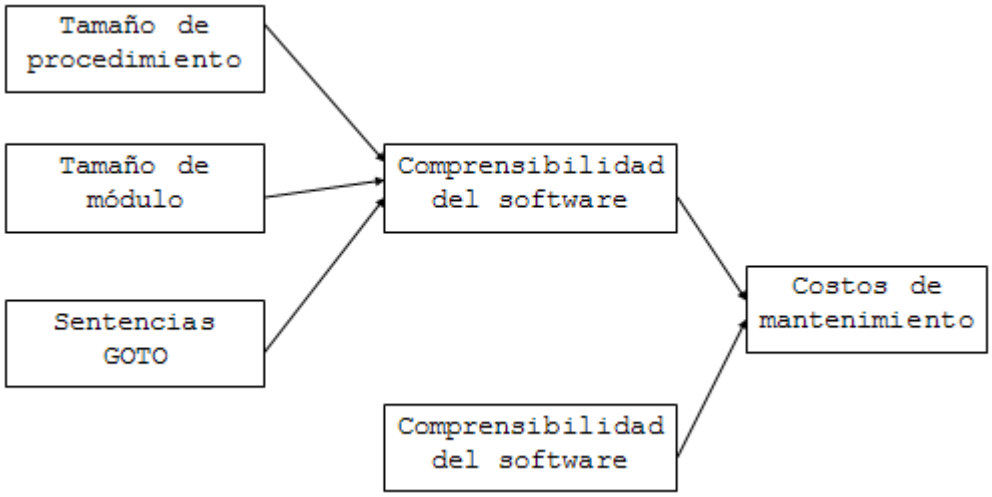
• D. Banker [BANK93] realiza un estudio basándose en tres métricas de complejidad del software:

- Tamaño de los procedimientos (PROCSIZE)
- Tamaño de los módulos (MODLSIZE)
- Densidad de sentencias GOTO (GOTOFAR)

Estas métricas están elaboradas considerando la siguiente estructura jerárquica del software:



El siguiente esquema expresa el modelo propuesto por Banker:



Banker realiza un estudio sobre 65 proyectos, obteniendo los siguientes resultados relativos a la comprensibilidad del software:

Variable	Valor medio	Desv.Estan.	Mínimo	Máximo
MODLSIZE	681	164	382	1104
PROCSIZE	43	18	13	87
GOTOFAR	0,024	0,016	0,0	0,07

Considerando estas variables y algunas otras medidas del tamaño y entorno del proyecto (agrupamos estas últimas en la variable OTROSFAC), construye un modelo estadístico descrito por:

$$HORAS = OTROSFAC + C_1 * SLOC * PROCSIZE + C_2 * SLOC * PROCSIZE^2 +$$

$$C_3 * SLOC * MODLSIZE +$$
$$C_4 * SLOC * MODLSIZE^2 +$$
$$C_5 * SLOC * GOTOFAR + \varepsilon$$

donde:

HORAS: Costo total de la etapa de mantenimiento

SLOC: número de líneas de código fuente añadidas o cambiadas en el mantenimiento.

Tras un análisis de regresión de los datos, se obtienen los siguientes coeficientes (c_i)

Variable	Coeficiente (c_i)
SLOC*PROCSIZE	-.0106
SLOC*PROCSIZE^2	.00012
SLOC*MODLSIZE	-.00011
SLOC*MODLSIZE^2	-4.4E-10
SLOC*GOTOFAR	1.317

Obteniendo la medida del esfuerzo (HORAS) con los valores medios y comparándola con la medida con los valores de máxima desviación tenemos los siguientes resultados:

Variable	% Impacto sobre el total
PROCSIZE	20 ó 25 %
MODLSIZE	10 %
GOTOFAR	12 %

- Stark y Vowell [STAR94] proponen un conjunto de 13 métricas para determinar la productividad en la etapa de mantenimiento, de las cuales, 5 están orientadas a medir la mantenibilidad del producto:

  – Grado de utilización de los recursos del computador
  – Tamaño del software
  – Densidad de fallos
  – Volatilidad del software
  – Complejidad del software

2.1.5 COMPRENSIBILIDAD: PRINCIPAL COMPONENTE DE LA MANTENIBILIDAD

Como vemos en el apartado anterior, Banker [BANK93] centra su modelo en la comprensión del software, que junto

con otros factores del proyecto determinan los costos en la etapa de mantenimiento.

Mendes-Moreira y Davies [MEND93] exponen la importancia de la comprensibilidad del software cuando dicen que se trata de una de las actividades que más recursos consume en los centros de proceso de datos. En dicho artículo corrobora lo dicho por N. Chapin [CHAP88] al enfatizar que la comprensión del sistema existente debe ser la primera fase en el ciclo de vida del mantenimiento. Sin una completa comprensión del software es imposible realizar un correcto análisis del impacto del cambio, es decir, de las repercusiones del cambio sobre el software y su documentación.

2.2 OTROS ASPECTOS INCIDENTES EN LA PRODUCTIVIDAD

Seguidamente se comentan aquellos elementos de la actividad de programación que consideramos que tienen mayor influencia sobre la productividad. Dejamos conscientemente fuera otros factores también importantes como reusabilidad, lengauajes de programación [PRES92, BERN94], etc.

2.2.1 MODELOS DEL CICLO DE VIDA

El ciclo de vida usado para desarrollar un programa es importantísimo como razona A. Velázquez en [VELA90]: “Si el programa a desarrollar es relativamente pequeño, puede realizarse con éxito sin más método que pensarlo, escribirlo y depurarlo. Sin embargo, un programa así desarrollado suele presentar dificultades de corrección y completitud funcional, prueba y mantenimiento, resultando un método impracticable en un proyecto grande.”

“Se hace necesario modelar el proceso de desarrollo de software de modo que se identifiquen las distintas tareas, su orden de realización y criterios de paso de una tarea a otra. Al conjunto de etapas por las que pasa un proyecto de software, desde que surge la necesidad del producto hasta que éste se desecha, se le da los nombres de ‘ciclo de vida del software’ o ‘proceso del software’”.

Realicemos pues una revisión de los que consideramos principales modelos de ciclo de vida de un proyecto de software.

Modelo en cascada

Es el modelo clásico del ciclo de vida de un proyecto software, que se basa en una construcción secuencial del software. Veamos sus principales ventajas y sus inconvenientes (según A. Velázquez en [VELA90]):

- El desarrollo de software se realiza alcanzando subobjetivos en un orden predeterminado (especificación, diseño, etc.)
- Las transiciones entre etapas sirven como puntos de control del progreso realizado y de los productos intermedios desarrollados.
- Reconoce la importancia del análisis y diseño previo a la codificación.

Las críticas recibidas son las siguientes (indicadas por Agresti en [AGRE86]):

- Resulta difícil separar el “qué” de la especificación del “cómo” del diseño [SWAR82]; además, es normal modificar las especificaciones según avanza el diseño, lo cual no aparece en el modelo.
- Es un modelo demasiado rígido que no engloba todas las situaciones posibles; frecuentemente el progreso no es secuencial sino iterativo, o el orden de algunas tareas cambia.
- Restringe la presencia del usuario a la especificación inicial, impidiendo su implicación en otras etapas del desarrollo.
- Las nuevas técnicas (p.ej. prototipado) e instrumentos de desarollo de software (p.ej. entornos CASE) que han aparecido no encajan en el modelo o fuerzan a hacer varias etapas simultáneamente.
- No recoge la gestión y coordinación del proyecto (la coordinación técnica se conoce como programación entre muchos, “programming-in-the-many”), que es fundamental en proyectos grandes.

No obstante, ha de considerarse otra ventaja que es su sencillez, lo que puede hacerlo apto como modelo de apoyo para realizar aportaciones al campo de los proyectos software que no dependan del modelo elegido.

Modelo con prototipos

Con este modelo [AGRE86], en los inicios del ciclo de vida ya se dispone de un modelo operativo del sistema que simula su comportamiento externo, lo cual permite depurar los requerimientos del usuario. Normalmente dicho prototipo

se realiza con lenguajes de cuarta generación (4GL), es decir, empleando instrucciones de alto nivel, más potentes y más próximas al lenguaje humano, aunque normalmente son poco eficientes.

Un prototipo del sistema, en manos del usuario, sirve para determinar qué requerimientos no están contemplados en el mismo o cuáles no se corresponden con la realidad. De este modo, los requerimientos pueden ser definitivos antes de comenzar el desarrollo real del producto, lo cual afecta de manera importante a la productividad global del proyecto.

Modelo transformacional

Tal vez el principal reto que presenta la ingeniería del software es la automatización del proceso de producción. El principal inconveniente se presenta en la primeras etapas del ciclo de vida ya que resultan más difíciles de automatizar. El modelo transformacional enfrenta este reto, cuyo esquema se presenta en la Figura 5 [BALZ83].

A. Velázquez [VELA90] explica así el funcionamiento del modelo: “El modelo toma las necesidades del usuario como punto de partida que, mediante un análisis de necesidades, servirá para el desarrollo de una especificación formal. Esta debe recoger las necesidades de una manera precisa (para representar exactamente el programa) y ser comprensible (para que el usuario pueda validarla). Entonces se va transformando la especificación paso a paso, hasta obtener un programa correcto y eficiente. Así pues, la implementación es un proceso iterativo de transformación de programas, en el que interviene el ordenador”.

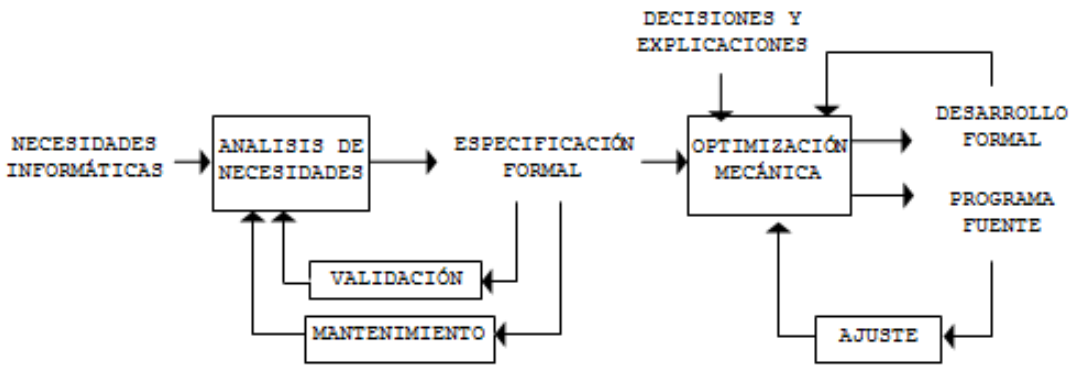


Figura 5.- Programación transformacional

Las ventajas de este modelo con respecto al modelo clásico son las siguientes:

a) Las inconsistencias o faltas de especificación pueden detectarse en los inicios del proceso, ya que se posee una especificación formal validable (y a veces ejecutable) de las necesidades informáticas.

b) Puesto que las transformaciones aplicadas hasta llegar al producto final son correctas se tiene que el programa final es correcto (responde a la especificación inicial de necesidades) y libre de errores. Es decir, no se necesita que el programa pase por una etapa de pruebas.

c) El mantenimiento es más fácil de realizar por dos motivos:

- El mantenimiento se realiza sobre la especificación formal de las necesidades (y no sobre el código) lo cual simplifica esta tarea ya que no se interfiere con los aspectos de diseño.
- El proceso transformativo genera documentación sobre el diseño en forma de un registro de desarrollo. De esta manera, tras realizar los cambios en la especificación, pueden reimplementarse todos estos aspectos de diseño a partir de dicho registro.

Este modelo también presenta algunos inconvenientes:

a) Falta de disponibilidad de criterios de evaluación. Existen grandes dificultades para establecer dichos criterios [HEND87].

b) Es un modelo relativamente “joven”, y su grado de desarrollo actual no permite utilizarlo de una manera extensiva.

c) Exige un alto grado de formalización que no siempre puede mantenerse, bien por la naturaleza de la aplicación o bien por la escasa formación de los programadores.

Modelo espiral

Modelo desarrollado por Boehm [BOEH88], que se basa en una serie de iteraciones en las que se cumplen unos objetivos definidos previamente. Este modelo está fundamentalmente orientado a disminuir riesgos y costes.

Este modelo considera que cada etapa del ciclo de vida del proyecto constituye un ciclo en una espiral que

representa la vida del proyecto. Cada etapa o ciclo consta de una secuencia de fases, entre las que cabe destacar, la especificación inicial de objetivos, un análisis de riesgos y una fase final de verificación de los resultados en confrontación con los objetivos previamente establecidos.

Es un modelo bastante estructurado, lo cual facilita la implantación de un modelo de productividad. Se contempla la posibilidad de construir prototipos en cada ciclo, con objeto de evaluar los resultados obtenidos antes de pasar a la siguiente etapa.

2.2.2 HERRAMIENTAS Y ENTORNOS DE PROGRAMACIÓN

Boehm [BOEH81] estima en un 10% el ahorro que supone el empleo de herramientas durante el desarrollo de un producto software. Pero, tal vez más importante que dicho ahorro, es el beneficio que el uso de herramientas reporta sobre el software.

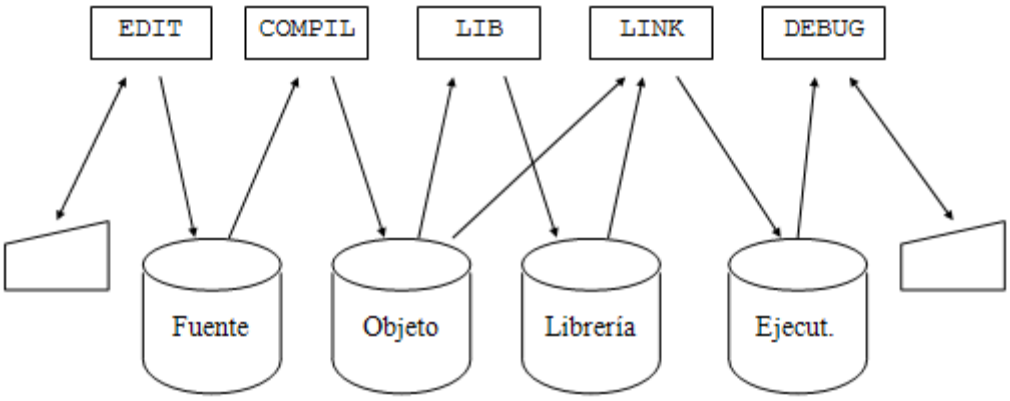
La etapa de **mantenimiento** es de las más beneficiadas por el uso de herramientas. Durante esta etapa existe un grave problema: la interpretación correcta de los contenidos del software. En la medida en que las herramientas favorecen la estandarización y homogeneidad del software, tendremos una mejor y más rápida interpretación del software y por tanto una reducción del esfuerzo de mantenimiento. Otros beneficios adicionales son:

- El software generado mediante herramientas se mantiene a través de lenguajes o útiles de un nivel superior.

- Existe un importante aumento de la calidad, ya que, al reducir la intervención humana se reducen también los errores humanos.

En un **entorno de programación clásico**[1], se dispone de un conjunto de herramientas preparadas para el proceso por lotes, que parten de ficheros de programas fuentes para dar como resultado programas ejecutables (Vease Figura 6).

[1]Aquí la palabra “entorno” se emplea con el significado “conjunto de facilidades o herramientas”. No obstante, esta palabra es más adecuada para los enfoques modernos, donde no se trata de un simple conjunto de facilidades, sino de un conjunto integrado con un único “front-end” de cara al usuario.



Edit .- Edición del programa
Compil .- Compilación o traducción
Lib .- Manejador de librerías
Link .- Montador de enlaces
Debug .- Depurador

Figura 6.- Entorno de programación clásico

Los "entornos" clásicos de programación son cada vez menos usados, sustituidos por los modernos **entornos integrados** en los que existe un único "front-end" con el usuario que controla el uso de todas las herramientas (véase Figura 7). Así, en su actividad cotidiana, el usuario no tiene que ser consciente de la existencia de las herramientas, ya que, muchas de ellas son utilizadas de forma transparente al usuario. Otras ventajas de los entornos integrados son:

- El trabajo es mucho más cómodo ya que se realizael entorno proporciona el acceso a las herramientas mediante menús.

- Se dispone de elementos de configuración que permiten adaptar el "front-end" a las necesidades del usuario (en mayor o menos medida según el entorno).

- Se elimina la necesidad de ficheros temporales para comunicar las distintas herramientas.

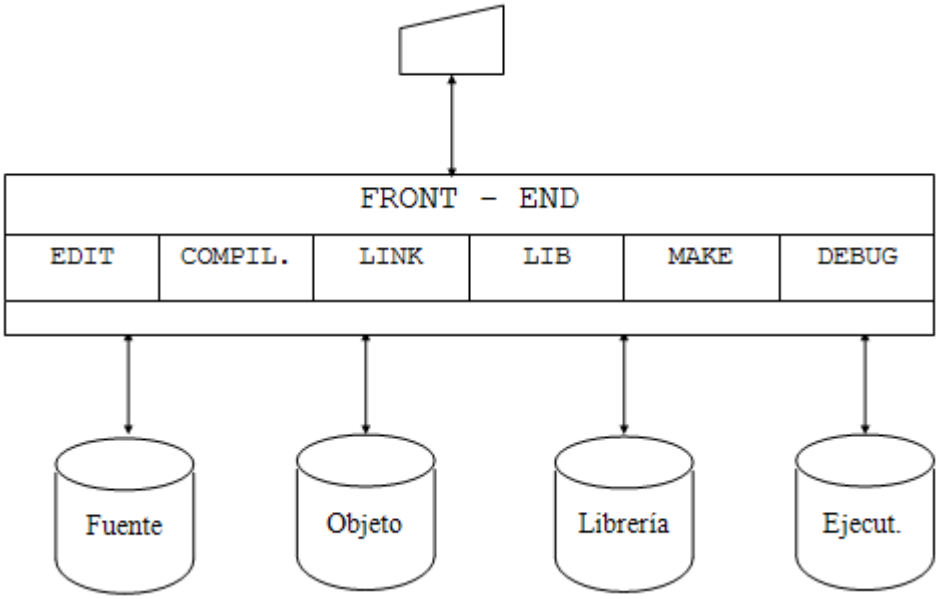


Figura 7.- Entornos de programación integrados

2.3 GRUPOS DE DECISIÓN COLECTIVA

Como ya se ha expresado, la mantenibilidad es un factor a tener muy en cuenta en los estudios de productividad. En el proceso de evaluación de este factor (o cualquier otro factor de calidad del software) aparece el problema de la subjetividad, es decir, las decisiones son tomadas por personas que pueden estar influidas por multitud de circunstancias y sus criterios de decisión pueden ser muy personales y poco objetivos. Para evitar dicha subjetividad se suelen emplear técnicas de decisión colectiva.

2.3.1 TÉCNICAS DE DECISIÓN COLECTIVA

Las técnicas de decisiones colectivas más extendidas son, sin duda, las técnicas de Delphi. Se trata de un conjunto de procedimientos desarrollado en la Rand Corporation, en 1948, con objeto de aprovechar al máximo los beneficios del consenso de expertos evitando los problemas causados por las reuniones en grupo.

Estas técnicas pueden aplicarse en el proceso de ingeniería del software para tomar cualquier decisión que sea susceptible de subjetividad. Un caso típico sería la estimación de costes del software. Las etapas de este proceso son las siguientes:

1) Un coordinador distribuye entre los estimadores (expertos) un documento de definición del sistema y un formulario para registrar la estimación del coste.

2) Cada estimador estudia la definición del sistema y cumplimenta su formulario anónimamente. Pueden plantear cuestiones al coordinador, pero no pueden discutir aspectos del sistema con otros estimadores.

3) El coordinador prepara y distribuye un sumario con las respuestas de los estimadores, e incluye las cuestiones destacadas por cada estimador.

4) Los estimadores completan otra estimación, de nuevo anónimamente, usando los resultados de la estimación previa. Los estimadores cuyos resultados difieran notablemente del grupo, pueden ser interrogados por separado sobre la cuestión.

5) El proceso se repite tantas veces como sea preciso hasta obtener una solución que el coordinador considere satisfactoria (en la que apenas haya disensiones). Durante este proceso no se realizan discusiones de grupo bajo ninguna circunstancia.

2.3.2 INTEGRACIÓN DEL G.D.C. EN EL MODELO DE CICLO DE VIDA

Diversas experiencias consideran la intevención de grupos de decisión colectiva con juicio de expertos en los procesos de estimación: el modelo COCOMO de Boehm [BOEH81] (vease apartado 2.1.1), el proyecto SPEM [SPEM88, SPEM90A, SPEM90B] (vease apartado 2.1.2), etc.

La integración de este instrumento de toma de decisiones en el ciclo de vida de los proyectos software ha sido ampliamente abordada por J.C. Granja [GRAN92/1, GRAN92/2].

El modelo propuesto por Granja (véase Figura 8), concebido para actividades de garantía y control de calidad, es adecuado para la toma de decisiones que requieran tal juicio de expertos. Este modelo conecta el equipo de producción con dicho grupo de expertos, y describe las actividades precisas para llevar a cabo las funciones de tal grupo.

El elemento central del modelo se denomina Grupo de Control de Calidad (GCC) que es una entidad compuesta a su vez por dos elementos:

a) Un elemento operativo: El **Equipo de Dirección del Programa de Revisión y Análisis de la Información** (EDPRAI).

b) Un **Elemento de Decisión en Primera Instancia** (EDPI).

Estos dos elementos interactúan y llegan finalmente a obtener un consenso de las opiniones de los expertos, siguiendo una serie de pautas basadas en la conocida técnica DELPHI de toma colectiva de decisiones, que, como se observa en la Figura 8, solo serán efectivas una vez que la dirección del proyecto las valide.

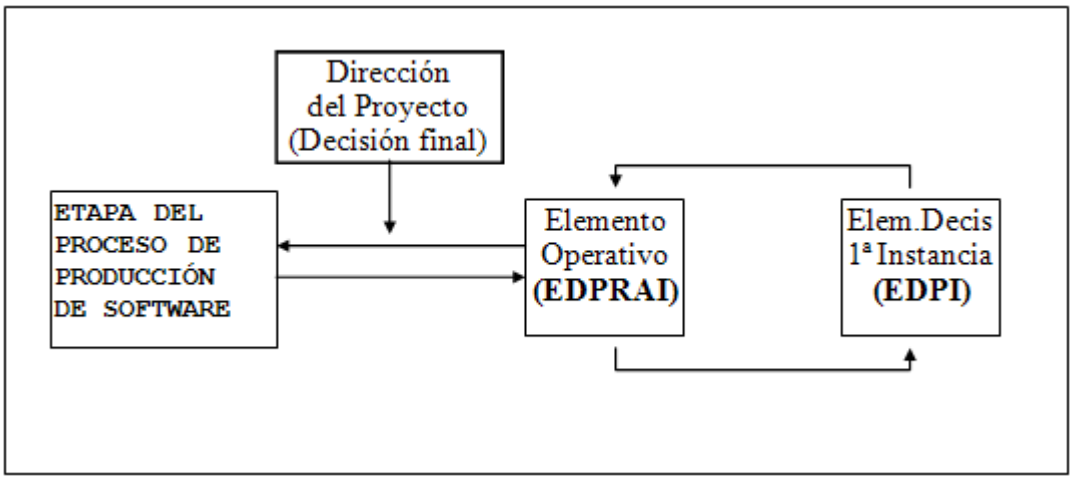


Figura 8.- Modelo de control de calidad

Descripción del EDPRAI

Este elemento operativo está compuesto por personas con experiencia en ingeniería del software, capacidad suficiente en estadística descriptiva y conocimiento en profundidad del proceso de producción. Sus funciones son:

- Recoger información del proceso de producción y adaptarla para su estudio por parte del EDPI.
- Establecimiento de una estructura de la documentación adecuada para consultarla sin dificultad en caso de posible auditoría.
- Dirigir y controlar el proceso de opinión del EDPI.
- Asesorar a la dirección y al resto de elementos de la organización del proyecto en temas referentes a la calidad.
- Realizar una labor de investigación para desarrollar estándares de calidad.
- Servir de nexo de unión entre el proceso de producción y el EDPI.

- Servir de nexo de unión entre el usuario y la organización de software.

Descripción del EDPI

Es un grupo de personas con una amplia experiencia en ingeniería del software y calidad del software. Deben ser voluntarios y sin conocimiento de la identidad del resto del grupo. Su función es contestar los cuestionarios suministrados por el EDPRAI siguiendo la normas que explícitamente se indiquen en ellos.

Método de trabajo

La metodología de trabajo consta de cinco etapas (aunque podrían ser ampliadas si así lo exige la complejidad del problema).

Etapa preparatoria

El EDPRAI envía la información pertinente a los expertos del EDPI con los problemas detectados y sus antecedentes. Los expertos devuelven, en el tiempo indicado por el EDPRAI, una lista con las principales áreas de interés relacionadas con el problema. El EDPRAI consolida y edita las áreas identificadas.

1ª Etapa

El EDPRAI elabora y envía cuestionarios para que los expertos den su opinión sobre los problemas detectados. Las predicciones de los expertos se tabularán por el EDPRAI que obtendrá una opinión media.

2ª Etapa

Cada experto recibe información estadística sobre las conclusiones de la primera etapa. Se pide a los expertos que revisen su opinión, y en caso de diferir de la opinión media, deberán facilitar las razones de su desacuerdo. El EDPRAI vuelve a resumir estadísticamente los resultados.

3ª Etapa

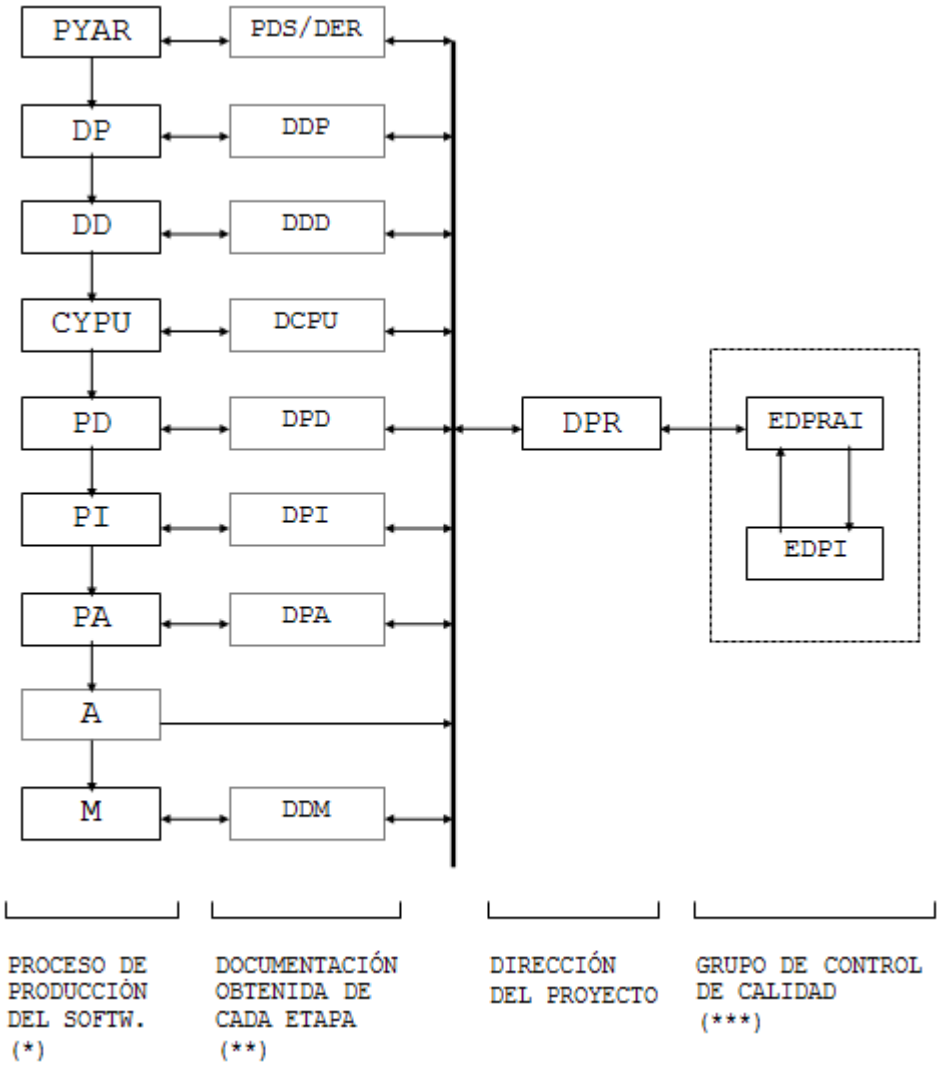
Basándose en los resultados de la segunda etapa, se vuelve a pedir a los expertos que revisen sus opiniones.

4ª Etapa

Se resumen estadísticamente los resultados de las tres etapas, se adjuntan los contraargumentos y se recircula todo una vez más para obtener predicciones finales. Con este resultado, el EDPRAI elabora el documento final determinando los procedimientos a llevar a cabo. Este documento se

enviará a la Dirección del Proyecto que se encargará de su examen, y si procede, a su implantación. En otro caso, se devolverá al EDPRAI para una nueva revisión.

El siguiente esquema muestra una visión general de la integración del modelo de control de calidad en el ciclo de vida de un proyecto software, según propone Granja.



(*) PROCESOS DE PRODUCCIÓN DEL SOFTWARE.

PYAR	PLANIFICACIÓN Y ANÁLISIS DE REQUERIMIENTOS.
DP	DISEÑO PRELIMINAR.
DD	DISEÑO DETALLADO.
CYPU	CODIFICACIÓN Y PRUEBA DE UNIDAD.
PD	PRUEBA DE DESARROLLO.
PI	PRUEBA DE INTEGRACIÓN.
PA	PRUEBA DE ACEPTACIÓN.
A	MOMENTO DE ACEPTACIÓN.
M	MANTENIMIENTO.

(**) DOCUMENTACIÓN RESULTANTE.

PDS/DER PLAN DE DESARROLLO DEL SOFTWARE Y Y DOCUMENTO DE ESPECIFICACIÓN DE REQUERIMIENTOS.
DDP DOCUMENTO DE DISEÑO PRELIMINAR.
DDD DOCUMENTO DE DISEÑO DETALLADO.
DCPU DOCUMENTO DE LA CODIFICACIÓN Y LA PRUEBA DE UNIDAD.
DPD DOCUMENTO DE LA PRUEBA DE DESARROLLO.
DPI DOCUMENTO DE LA PRUEBA DE INTEGRACIÓN.
DPA DOCUMENTO DE LA PRUEBA DE ACEPTACIÓN.
DDM DOCUMENTO DEL DESARROLLO DEL MANTENIMIENTO.

(***) GRUPO DE CONTROL DE CALIDAD.
EDPRAI EQUIPO DE DIRECCIÓN DEL PROGRAMA DE REVISIONES Y ANÁLISIS DE LA INFORMACIÓN.
EDPI ELEMENTO DECISOR EN PRIMERA INSTANCIA.

La primera columna (*) del esquema de ciclo de vida muestra la secuencia de las etapas de producción del software, desde la planificación y análisis de requerimientos hasta el mantenimiento.

La segunda columna (**) representa los elementos de información que se van a tomar de cada etapa del proceso para el control de la calidad.

La tercera columna expresa la necesidad de validar, a través de la dirección del proyecto, cualquier decisión tomada por el grupo de control de calidad.

La cuarta columna (***) representa el grupo de control de calidad formado por los dos elementos ya comentados (EDPRAI y EDPI).

3 ESTUDIOS PREVIOS DE PRODUCTIVIDAD DEL SOFTWARE

La línea de investigación seguida en el transcurso de de esta tesis, hasta llegar al modelo de productividad propuesto, pasa por algunos estudios previos con interesantes incursiones en este campo. Es por ello que, antes de exponer la que consideramos nuestra mayor contribución al área de estudio, se van a presentar tres trabajos sobre la productividad del software que han sido fruto de la investigación que conduce a esta tesis.

El primero de ellos (apdo. 3.1), publicado en [BARR94] y defendido en el “Third International Conference on Software Quality Management SQM’95” en [BARR95], surge de la propia experiencia del doctorando en una empresa de desarrollo de software, ante la problemática ocasionada por el control de versiones durante la etapa de mantenimiento, en un mercado de clientes con una diversidad importante de necesidades de mantenimiento[2] y niveles económicos. Este aspecto motiva la necesidad de ofertar varios tipos de contrato de mantenimiento, con objeto de que el cliente se acoja a uno u otro según sean sus intereses. Esto constituye una fuente de problemas en cuanto a control de versiones del software se refiere, provocando pérdidas importantes de productividad en las actividades de mantenimiento.

Al considerar la estrecha relación que existe entre calidad del software y productividad en la etapa de mantenimiento, se realiza un estudio de los factores de calidad que un producto software debe poseer (apdo. 3.2), sin olvidar el hecho de que la implantación de un sistema de calidad implica un inversión económica importante. En la búsqueda del conjunto de factores de calidad más idóneo, se considera que el usuario tiene mucho que decir en cuanto a la calidad del producto que desea. Por tanto, se hace un estudio de dicho conjunto de factores partiendo de la ERCU (Especificación de Requerimientos de Calidad del Usuario) [GONZ94].

Continuando con el estudio de las métricas de calidad del software y su conexión con la productividad, se propone una técnica de análisis de la calidad del software (apdo.

[2]Las necesidades de mantenimiento de un usuario es un concepto subjetivo que el propio usuario determina considerando sus perspectivas de crecimiento, necesidades de adaptación a un entorno cambiante, etc. El contraste de sus necesidades con los tipos de contrato ofertados será un elemento de juicio para decidirse por uno u otro tipo.

3.3), publicada en [MART94], que partiendo de varias métricas ofrece una medida global de la misma, al tiempo que determina cuál es el factor crítico que más afecta a la calidad y por tanto a la productividad. Se propone el empleo de una herramienta gráfica para la representación de las métricas, y así obtener una visión global y rápida de la calidad, lo cual permitirá realizar comparaciones en una forma ágil entre distintos proyectos o entre las fases de un mismo proyecto.

3.1 PRODUCTIVIDAD EN EL CONTROL DE VERSIONES

Una visión del problema de la productividad en la etapa de mantenimiento se presenta en los sistemas informáticos de propósito comercial, es decir, en aquellos proyectos no realizados "a medida" de un único usuario, sino con vistas a su venta a múltiples usuarios [BARR94, BARR95].

El problema surge ante la imposibilidad de realizar de forma eficiente el control de versiones en un *entorno de mantenimiento selectivo*[3], lo que afectará de forma negativa e importante en la productividad.

Un principio básico que debe ser observado en el proceso de control de versiones se conoce como *principio de eliminación de copias* [PLAI93], y dice: "Si un componente permanece invariable en un cambio de versión, entonces no se debe crear una nueva copia de este componente para la nueva versión." El incumplimiento de este principio daría lugar a copias idénticas de componentes con la consiguiente multiplicación de esfuerzo en la realización de cambios.

Para observar este principio es preciso convenir cierto criterio de selección de la versión de cada componente que deba ser elegida para construir una versión del sistema. El criterio utilizado se denomina **CMR** o **"Componente Más Relevante"**, ya introducido en [PLAI93]. Decimos que las versión *V* de un componente es más relevante que otra *W*, y lo representamos como $W \leq V$ si y solo si existe una secuencia de cambios que transforma la versión W

[3]**Entorno de mantenimiento selectivo** es un término utilizado para denotar un entorno de mantenimiento en el cual el usuario tiene la facultad de determinar qué derechos de mantenimiento desea contratar. Se ofrecen tres tipos:
Tipo 1.- Con derecho al mantenimiento correctivo
Tipo 2.- Con derecho a mantenimiento correctivo y adaptativo
Tipo 3.- Con todos los derechos de mantenimiento

en la versión *V*. Así, el componente más relevante respecto a una versión *V* será:

$$CMR(V) = \max_{\leq}(W / W \leq V)$$

Como se comprende fácilmente, para asegurar la existencia de uno y solo un componente más relevente para cada versión V, **la relación $\leq$ deberá ser total**. De otro modo, podríamos tener un conjunto de máximales o ninguna versión del componente que cumpla $W \leq V$.

Planteamiento del problema

En este contexto surgen tres problemas:

a) Cuando realizamos un cambio, debemos saber qué versiones de cada componente deben ser consideradas (es decir, qué versiones de los componentes están “vivas” en cada momento).

b) Cuando realizamos un cambio, debemos conocer qué versiones del sistema deben ser consideradas (es decir, qué versiones del sistema están “vivas” en cada momento).

c) Cuando construimos una determinada versión del sistema, debemos saber qué versión de cada componente debe ser escogida para la construcción.

El modelo clásico de control de versiones no resulta apropiado para solventar estos problemas en un entorno de mantenimiento selectivo, observando el principio de eliminación de copias. Los siguientes ejemplos tienen por objeto argumentar esta afirmación de una forma clara.

Ejemplo 1:
Sea el sistema
$S = \{ a, b, c, d \}$

Sea la siguiente secuencia de cambios
$X = [x_1, x_2]$
con los siguientes componentes afectados en cada cambio:
$S/x_1 = [b,c], \quad S/x_2 = [a,c]$

El siguiente esquema muestra la evolución del sistema especificando el modo de construcción de cada versión, siguiendo el criterio de construcción CMR ya comentado.

$$S_0 \xrightarrow{x_1} S_1 \xrightarrow{x_2} S_2$$
$$a_0 = a_0 \rightarrow a_2$$
$$b_0 \rightarrow b_1 = b_1$$
$$c_0 \rightarrow c_1 \rightarrow c_2$$
$$d_0 = d_0 = d_0$$

Como se observa, el sistema evoluciona de forma lineal desde la versión 0 a la 1 y más tarde a la 2, y las versiones se construyen sin problema alguno respetando el principio de eliminación de copias. Así, el componente *a* no cambia con x_1 por lo que no se crea un versión a_1, sino que la versión S_1 del sistema utiliza la versión a_0 de dicho componente.

Este es el caso de que los dos cambios x_1 y x_2 sean del mismo nivel (vease definición de mantenimiento selectivo) o x_1 fuese de nivel inferior a x_2.

Ejemplo 2:

Si consideramos el mismo sistema del anterior ejemplo y los mismos cambios, pero siendo el cambio x_1 de mayor nivel que el cambio x_2, tendremos que es preciso crear una versión $S_{0.1}$ que no contenga el cambio x_1 pero sí el cambio x_2.

$$\begin{array}{l} S_0 \xrightarrow{x_1} S_1 \xrightarrow{x_2} S_2 \\ \downarrow x_2 \\ S_{0.1} \end{array}$$

En este caso, el componente *a* en su versión a_2 y en su versión $a_{0.1}$ serían dos copias idénticas, y no podrían reducirse a una al no poder relacionar mediante el criterio CMR las versiones 0.1 y la versión 2. En este esquema de versiones, la relación *componente más relevante* antes expuesta deja de ser total, y por tanto, al no poder reducir a una única copia ambas versiones del componente *a* se viola el principio de eliminación de copias.

Solución propuesta

Se propone la representación del conjunto de versiones en un espacio de n dimensiones, donde n es el número de tipos de cambio, es decir,

V = (P,A,C)

siendo,

P = Cambio perfectivo
A = Cambio adaptativo

C = Cambio correctivo

Los cambios se identificarán mediante ternas <i,j,k>, siendo,
i = número de orden en la lista global
j = tipo de cambio (P, A o C)
k = número de orden en la lista parcial j

Ejemplo:
X = [<1,P,1> , <2,C,1> , <3,P,2>]

Para abordar los problemas planteados, es preciso definir las tres relaciones de orden siguientes:

Orden parcial (<P<)

$(x_1,x_2,...,x_n)$ <P< $(x_1',x_2',...,x_n')$ si y sólo si

$x_1 \leq x_1'$, $x_2 \leq x_2'$,..., $x_n \leq x_n'$

Orden total (<T<)

$(x_1,x_2,...,x_n)$ <T< $(x_1',x_2',...,x_n')$ si y sólo si

$x_1 < x_1'$ o
$x_1 = x_1'$, $x_2 < x_2'$ o
...
$x_1 = x_1'$, $x_2 = x_2'$, ..., $x_n < x_n'$ o
$x_1 = x_1'$, $x_2 = x_2'$, ..., $x_n = x_n'$

Orden de nivel J (<N_j<)

$(x_1,x_2,...,x_n)$ <N_j< $(x_1',x_2',...,x_n')$ si y sólo si

$(x_1,x_2,...,x_n)$ <T< $(x_1',x_2',...,x_n')$ y
$x_i = x_i'$ para todo i desde 1 hasta j-1

Basándonos en estas relaciones, podemos contruir los siguientes procedimientos para resolver cada uno de los problemas planteados:

a) Procedimiento de Obtención de Versiones de Componentes Afectadas (POVCA)

Sea el sistema $S = \{C_1,C_2,...C_n\}$

Sea $S/<i,j,k>$ el subconjunto de S con los componentes afectados por el cambio <i,j,k>

Para cada $C \in S/<i,j,k>$ tenemos que:

VC_{i-1}= {versiones del componente C, existentes antes de comenzar con el cambio i}

Las versiones del componente C afectadas por el cambio <i,j,k> vienen dadas por la siguiente expresión:

$$VC_{i-1}/<i,j,k> \;=\; \max_{<Nj<}(VC_{i-1}) \;=$$
$$= \{V' \in VC_{i-1} \,/\, no\,\exists\; V \in VC_{i-1} \;\; t.q.\; V' <_{Nj}< V\}$$

La lectura de esta expresión sería: las versiones del componente C afectadas por el cambio <i,j,k> son el conjunto de maximales, según el orden parcial $<N_j<$, de todas las versiones del componenten C en el momento del cambio.

b) Procedimiento de Obtención de Versiones del Sistema Afectadas (POVSA)

VS_{i-1}= {versiones del sistema, existentes antes de comenzar con el cambio i}

Las versiones del sistema afectadas por el cambio vienen dadas por la expresión:

$$VS_{i-1}/<i,j,k> \;=\; \max_{<Nj<}(VS_{i-1}) \;=$$
$$= \{V' \in VS_{i-1} \,/\, no\,\exists\; V \in VS_{i-1} \;\; t.q.\; V' <_{Nj}< V\}$$

La lectura de esta expresión sería: las versiones del sistema afectadas por el cambio <i,j,k> son el conjunto de maximales, según el orden parcial $<N_j<$, de todas la versiones del sistema existente en el momento del cambio.

c) Proceso de Construcción de Versiones del Sistema (PCVS)

El proceso de elección de las versiones de los componentes para construir una versión W del sistema (S_w) viene dado por:

$$C^W = \max_{<T<}\left\{V' \in VC \,/\, V' < P < W\right\}$$

donde:

C^w es la versión elegida de C para construir W

VC es el conjunto de versiones del componente C en el momento de construir la versión.

La lectura de esta expresión sería: la versión de un componente C elegida en la construcción de una versión W del sistema debe ser el máximo según el orden total <T< de todas

las versiones de C inferiores a W según el orden parcial <P<.

En **conclusión**, la solución aquí propuesta permite eliminar el riesgo de pérdidas de productividad que pueden venir motivadas por las copias idénticas de componentes software que se generan con un mal control de versiones en la etapa de mantenimiento.

3.2 ESTUDIO DE LOS FACTORES MÁS INFLUYENTES

Una de las primeras incógnitas en la implantación de un sistema de calidad consiste en la selección de los factores de calidad a controlar. Existen multitud de ellos, siendo bastante reconocidos los estudios al respecto de McCall [MCCA77], quien señala tres grupos: **operaciones del producto** (corrección, fiabilidad, eficiencia, integridad, facilidad de uso), **revisión del producto** (facilidad de mantenimiento, flexibilidad, facilidad de prueba) y **transición del producto** (portabilidad, reusabilidad, interoperatividad).

Considerando que el control de todos los factores puede resultar excesivamente costoso para muchas empresas de software, se propone un método de reducción de dicho conjunto en base a las Especificaciones de Requerimientos de Calidad del Usuario (ERCU) [GONZ94].

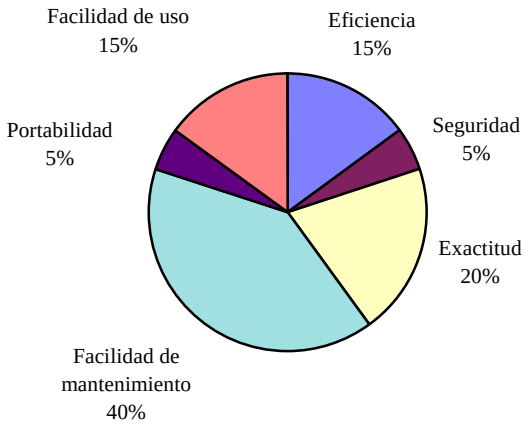
Para organizar dichas especificaciones se propone el uso de las propiedades no funcionales que define **EuroMétodo** [EURO94], es decir, aquellas que no determinan la funcionalidad del software sino otro tipo de características:

- Eficiencia
- Seguridad
- Exactitud
- Facilidad de mantenimiento
- Portabilidad
- Facilidad de uso

La ERCU deberá contener una valoración de cada propiedad no funcional. Sería preciso disponer de una correspondencia entre estas propiedades y los factores de calidad, con lo cual se tendría una valoración de esfuerzo a invertir en el control de cada factor.

Considerese que no se trata de dar una valoración cuantitativa de las necesidades de calidad del usuario, sino más bien, un análisis comparativo que indique el interés relativo de unas propiedades con otras desde el punto de

vista del usuario. Así, a cada propiedad no funcional se le va a asignar un porcentaje, que va a significar la porción del esfuerzo total de aseguramiento y control de la calidad. Véase el siguiente ejemplo gráfico:



En la Figura 9 se expone la correspondencia entre tales propiedades del software y los factores de calidad de McCall, así como los argumentos que motivan tal relación.

- **Eficiencia**.- Como es evidente, se ha de corresponder con el factor de calidad de **eficiencia** que propone McCall.

- **Seguridad.-** Se corresponde en definición con el factor **integridad** propuesto por McCall.

- **Exactitud**.- Se entiende que cuando el usuario requiere exactitud está pidiendo no sólo que el programa satisfaga los requerimientos (**corrección**) sino también que lo haga de una forma precisa o fiable (**fiabilidad**).

- **Facilidad de mantenimiento**.- Al requerir esta propiedad se está considerando una importante probabilidad de cambios durante el mantenimiento. Por tanto, se requieren características de calidad que hagan fácil la tarea de realización de cambios: **facilidad de mantenimiento, flexibilidad y facilidad de prueba**.

- **Portabilidad**.- Este requisito del usuario se corresponde, como es lógico, con el factor **portabilidad** de McCall, aunque también puede implicar la **interoperatividad**, es decir, la facilidad de acoplar el sistema a otro.

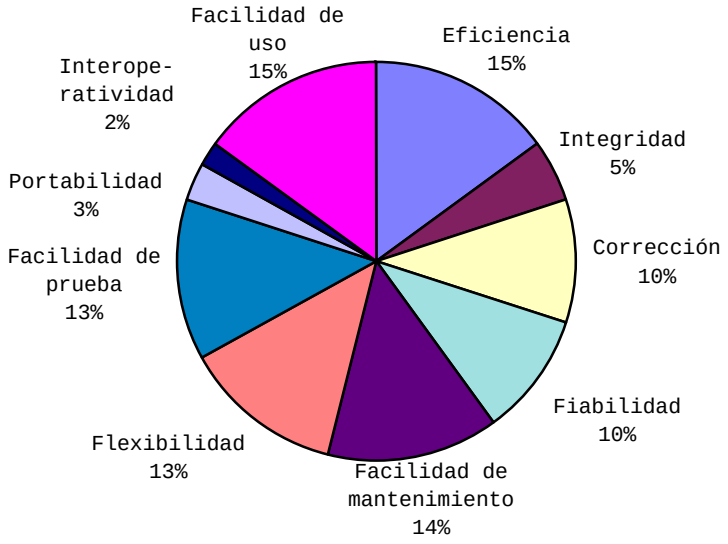
- **Facilidad de uso**.- Esta propiedad se corresponde, de una forma lógica, con el factor **facilidad de uso** de McCall.

Eficiencia
Seguridad
Exactitud
Facilidad de mantenimiento
Portabilidad
Facilidad de uso

Corrección				X		
Fiabilidad				X		
Eficiencia						X
Integridad					X	
Facilidad / uso	X					
Facilidad / mto.			X			
Flexibilidad			X			
Facilidad prueba			X			
Portabilidad		X				
Reusabilidad						
Interoperatividad		X				

Figura 9.- Propiedades del software vs. factores de calidad según McCall.

Siguiendo con el ejemplo, al considerar la correspondencia entre las propiedades no funcionales y los factores de calidad tendremos la siguiente distribución (repartiendo de forma equitativa entre los factores relacionados con una misma propiedad):



En conclusión diremos que, las especificaciones de requerimientos de calidad del usuario (ERCU) tienen una gran importancia a la hora de establecer un plan de calidad en un proyecto software, en tanto en cuanto van a proporcionar una valoración comparativa de los factores de calidad a tener en cuenta. Esto va a permitir equilibrar los esfuerzos en el

aseguramiento y control de la calidad, evitando inversiones desmedidas en aspectos de calidad poco importantes para el usuario.

3.3 ESTUDIO GRÁFICO DE LAS MÉTRICAS DE CALIDAD

Este trabajo [GONZ94] está basado en el modelo de control de calidad planteado por J.C. Granja en [GRAN92/2] (vease apdo. 2.3.2). Consiste en un estudio gráfico de los factores de calidad del software, como un intento de conjugar un conjunto de métricas de calidad dando una visión global de la calidad del producto.

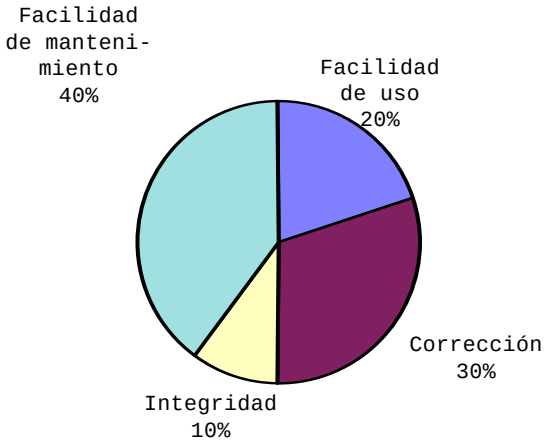
Este estudio constituye un complemento al anteriormente expuesto, ofreciendo una herramienta que proporciona gráficamente una medida de la calidad del producto. El equipo de control de calidad podrá hacer uso de esta herramienta para realizar un seguimiento de la evolución de la calidad del producto, o bien hacer comparaciones entre distintos proyectos.

La descripción de esta herramienta gráfica se expondrá apoyándose en un ejemplo, de forma que se pueda comprender mejor y más rápidamente.

El punto de partida viene dado por el conjunto de factores de calidad a considerar, con sus correspondiente valoraciones de importancia (vease apartado anterior).

Cada factor tendrá asociada una métrica que será utilizada en determinados instantes del proyecto para medir dicho factor de calidad.

Ejemplo:



Considérese que el rango de cada métrica es de 0 a 10, siendo 0 el mejor valor, 10 el peor valor y el valor objetivo igual a 5 en los cuatro casos, lo cual no constituye reducción en la generalidad de la herramienta propuesta. Por valor objetivo nos referimos al valor ideal de calidad, teniendo en cuenta que valores inferiores serán demasiado costosos de implementar y valores superiores implican una calidad deficiente.

La representación gráfica propuesta pondera cada medida de calidad con el orden de importancia de dicho factor de calidad.

Sea O_i el orden de importancia (medido en tanto por ciento) del factor *i* y *M* una medida tomada del factor *i.* Entonces, la medida ponderada *M'* vendrá dada como:

$$M' = M * O_i * N^oFactores$$

La representación gráfica consiste en un gráfico circular con las siguientes características:

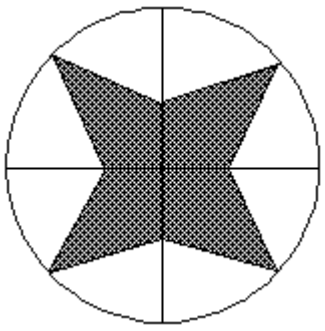
a) El círculo se divide en tantos sectores o áreas del mismo tamaño como factores de calidad intervienen en el estudio.

b) El centro del gráfico representa el mejor caso y el borde exterior el peor caso. Un círculo concéntrico representa las metas para cada factor de calidad o nivel objetivo de calidad.

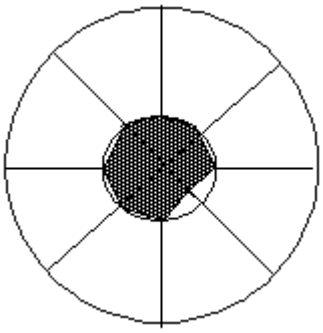
c) En la bisectriz de cada sector se traza un radio sobre el que se marcará el nivel de calidad medido de dicho factor.

d) El punto que representa la medida se une mediante dos líneas con las dos intersecciones del círculo objetivo y los lados del sector.

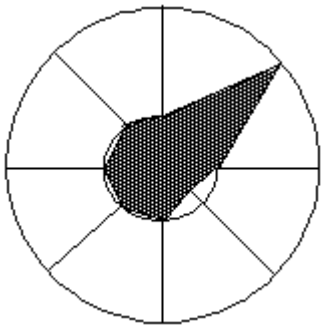
e) El polígono así construido ofrece una medida global de la calidad. Las siguientes figuras son características:



ESTRELLA: Calidad mejorable para todos los factores de calidad implicados en el producto.



OJO: Nivel satisfactorio o suficiente de calidad. Todos los factores cumplen el objetivo prefijado.



PUNTA DE FLECHA: Existe un factor crítico que disminuye la calidad total, mientras el resto de los factores ya han alcanzado sus valores objetivo.

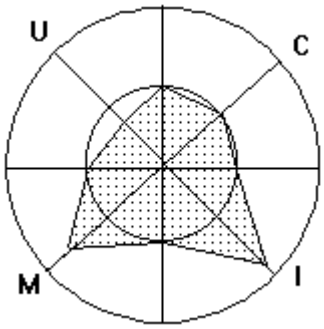
Supongamos que tenemos las siguientes medidas ponderadas tomadas en tres instantes del proyecto, listas para ser sometidas a la herramienta gráfica.

Fases de extracción de métricas

Factor	Fase 1	Fase 2	Fase 3
U: Facilidad de uso	3	5	6
C: Corrección	5	5	5
I: Integridad	9	8	6
F: Facilidad de mantenimiento	8	7	5

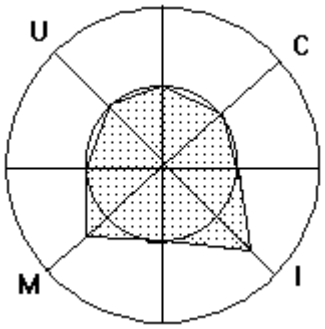
Tendríamos las siguientes gráficas para cada fase de medida:

Fase 1



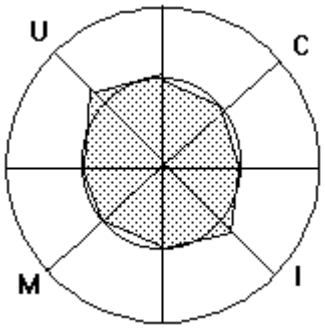
Los factores U y C están en unos valores adecuados, pero tanto el factor I como el M necesitan ser mejorados.

Fase 2



Se observa que el factor U ha empeorado, pero manteniéndose dentro del nivel objetivo de calidad. Los factores M e I han mejorado algo, pero no lo suficiente.

Fase 3



El nivel objetivo de calidad está prácticamente objetivo, salvo dos pequeñas desviaciones de un punto en los factores U e I.

Una rápida mirada a estas gráficas pone de manifiesto la evolución de calidad a lo largo del proyecto. Se observa que los factores I y M han sido mejorados de forma importante mientras que los factores U y C prácticamente se han mantenido en el valor objetivo.

4 DISEÑO DE UN MODELO DE PRODUCTIVIDAD

Partiendo de los estudios argumentados en los capítulos 1 y 2, pasamos a realizar nuestra aportación en un campo en el que, como hemos podido ver, no está suficientemente solucionado, mediante técnicas adecuadas, el problema de la evaluación y mejora de la productividad del software en la etapa de mantenimiento.

4.1 INTRODUCCIÓN AL DISEÑO DEL MODELO

Como ya se ha indicado en el capítulo 1, la elección del mantenimiento como etapa donde centrar el estudio de la productividad ha estado motivada por la enorme proporción de esfuerzo que la etapa de mantenimiento supone con respecto al resto del proyecto.

Por otro lado, la elección de un factor de calidad del software, la mantenibilidad, como determinante de la productividad, en lugar de otro tipo de factores que, sin duda, también influyen de manera importante según los estudios ya citados en capítulo 2, se ha realizado considerando las siguientes razones:

- Suele resultar más económico invertir en mantenibilidad que en cambiar herramientas, hardware o personal.

- Por muchas inversiones que se hagan en otras direcciones, si la mantenibilidad del software es deficiente, la productividad en el mantenimiento también va a ser deficiente.

La experiencia de Boehm en [BOEH79], ya relacionada en el capítulo 2, corrobora estas razones al expresar que puede haber una relación de 40 a 1 entre el esfuerzo de mantenimiento en situación de baja mantenibilidad y el esfuerzo en nuevos desarrollos, equiparable este último al esfuerzo en mantenimiento con alta mantenibilidad.

Por tanto, y según Boehm, la peor/mejor mantenibilidad del producto se traduce en una multiplicación/división del esfuerzo en mantenimiento, que puede llegar a ser del orden de varias decenas.

Considerando que los modelos actuales no abordan el problema de la evaluación de la productividad en el mantenimiento partiendo del factor mantenibilidad, se

propone un modelo de productividad que ofrece los procedimientos y técnicas necesarios para evaluar y mejorar la productividad a partir del estudio de dicho factor de calidad.

La evaluación de la productividad, como cualquier otro proceso de evaluación o estimación precisa de dos elementos fundamentales (como se señala en [GRAN95] pág. 3.2):

- **Datos históricos**
- **Juicio de expertos**

Los **datos históricos** reflejan experiencias similares en proyectos pasados y sirven de ayuda en las estimaciones.

Pero la interpretación de tales datos es algo muy susceptible a la subjetividad. Así, la estimación puede ser demasiado dependiente de la persona encargada de la misma. Por ello, es preciso disponer de algún elemento que elimine al máximo tal subjetividad.

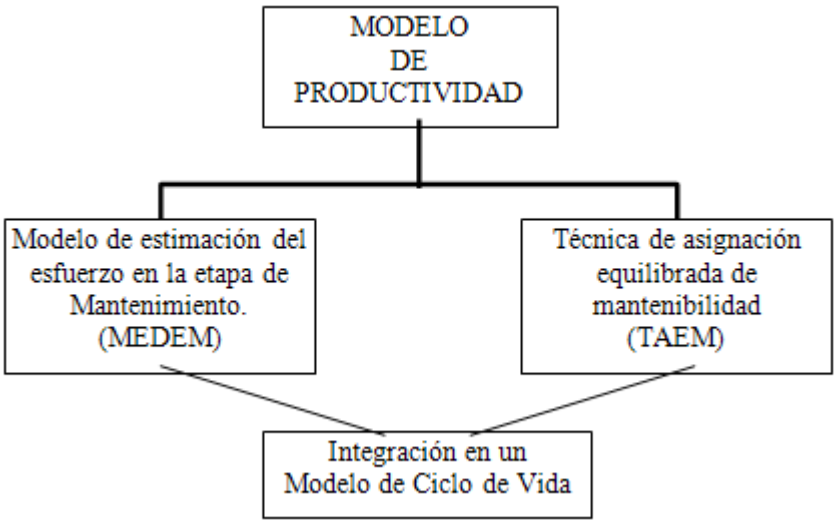
Juicio de expertos. El modelo propuesto por J.C. Granja (vease apdo. 2.3.2) hace una aportación en tal sentido y constituye la base de nuestro **grupo de decisión colectiva.** Se trata de un grupo con características muy similares al GCC de Granja, pero cuyas funciones no están restringidas únicamente a tomar decisiones referentes a la calidad del producto, sino mayormente a realizar estimaciones.

Estos dos elementos constituyen las herramientas básicas del modelo propuesto, cuyo objetivo, ya expresado al plantear el problema, se alcanzará mediante el estudio del factor de calidad que, como ya se ha argumentado, es el que más incide en la productividad de la etapa de mantenimiento: la **mantenibilidad**.

Una vez diseñado el modelo, se procederá a su integración en un entorno de programación, describiendo las herramientas precisas para tal propósito.

4.1.1 VISIÓN GLOBAL DEL MODELO DE PRODUCTIVIDAD

La contribución realizada en esta tesis, que en su conjunto denominamos “modelo de productividad”, está compuesta por tres componentes principales que se observan en el siguiente gráfico:



Estos tres componentes están tratados en detalle en los apartados 4.3, 4.4 y 4.5 respectivamente.

4.1.2 NOTACIÓN EMPLEADA

Hemos de hacer algunas consideraciones en cuanto a la notación empleada en el desarrollo del modelo. A continuación se relacionan algunos símbolos que empleamos:

KS Medida de tamaño, cuya unidad de medida es igual a *mil líneas de código* que también denotaremos como *k-líneas*.

MM Medida de esfuerzo, siendo la unidad de medida un hombre-mes.

Algunas normas de referentes a la terminología empleada son las siguientes:

Letras minúsculas.- Cuando un término de una expresión comienza con letra minúscula corresponde a una medida real tomada de datos históricos:

mm_x Medida de esfuerzo real tomada de los datos históricos.

ks_x Cantidad de k-líneas reales tomada de los datos históricos.

Letras mayúsculas.- Comienzan por letra mayúscula los demás términos, es decir, aquellos que no corresponden a medidas reales tomadas de datos históricos. Ejemplo:

MM_x Medida de esfuerzo estimada.

KS_x Cantidad de k-líneas estimada.

4.2 FUNDAMENTOS DEL MODELO

Considerando el primer objetivo de nuestra tesis, formulado en el apdo. 1.3, el modelo a diseñar debe solventar los dos problemas allí expuestos:

(a) Estimación de la productividad en el mantenimiento a partir de una medida de la mantenibilidad.

(b) Obtención de la asignación más adecuada de características de mantenibilidad.

Para tal propósito se precisa de varios elementos que van a constituir el fundamento de nuestro modelo:

1.- **Características de mantenibilidad.-** Es preciso distinguir las características funcionales del software de las no funcionales. La mantenibilidad constituye una de las últimas.

2.- **Medida de la productividad.-** En primer lugar se ha de establecer la medida o medidas de productividad a utilizar en el modelo.

3.- **Criterio de asignación de mantenibilidad.-** Es preciso determinar cuál es el criterio en base al cual se va a asignar más o menos mantenibilidad.

4.- **Tráfico de cambios anual (TCA).-** Se propone una clasificación del software basada en el TCA.

5.- **Métricas de tamaño y complejidad**.- El tamaño y la complejidad del software inciden poderosamente sobre la mantenibilidad o facilidad de mantenimiento. No obstante, son aspectos que también tiene una estrecha relación con la funcionalidad.

6.- **Métricas de mantenibilidad.-** Es necesario disponer de un conjunto de métricas de mantenibilidad apropiadas para el propósito del modelo.

4.2.1 CARACTERÍSTICAS DE MANTENIBILIDAD

Como se expresaba en el apdo. 3.2, el EuroMétodo distingue entre las propiedades funcionales y no funcionales del software. Una de estas propiedades no funcionales es la facilidad de mantenimiento o mantenibilidad.

Vamos a considerar dos tipos de características de mantenibilidad:

a) **Características fijas**.- Son características de las que depende la mantenibilidad, pero que todo módulo, sean cuales sean sus necesidades de mantenibilidad, debe mantener dentro de unos márgenes preestablecidos. Las consideradas en nuestro modelo son: el tamaño del módulo y su complejidad ciclomática. En el apartado 3.3.5 se comentan con más profundidad estas características.

b) **Características dependientes del nivel de mantenibilidad**.- Un módulo deberá mantener los valores de estas características dentro de unos rangos u otros, dependiendo de su nivel de mantenibilidad[4]. Como ya se dijo (véase Figura 2), la mantenibilidad viene dada por tres componentes: comprensibilidad, modificabilidad y testeabilidad, que van a ser, precisamente, las tres características dependientes del nivel de mantenibilidad que vamos a considerar. Más concretamente, se trata de medir:

- Autodocumentación[5] como principal característica determinante de la comprensibilidad.
- Generalización[6] como principal característica determinante de la facilidad de modificación.
- Instrumentación[7] como principal característica determinante de la testeabilidad.

En el apartado 3.3.6 se comentan con más profundidad estas características.

4.2.2 MEDIDAS DE PRODUCTIVIDAD

Las medidas de productividad pueden disponerse *a priori* o *a posteriori*, es decir, como una estimación de la productividad o como una medida la productividad real.

La estimación de productividad clásica y evidente (por definición) para tomar *a priori* viene dada como el cociente

[4]Entiéndase mayor o menor grado de mantenibilidad. Más tarde se definirá con mayor precisión este concepto.

[5]Grado en que el código fuente proporciona información significativa.
[6]Grado en que se han extraido del programa las constantes y las limitaciones de expansión.
[7]Grado en que el programa muestra su propio funcionamiento identificando los errores que aparecen.

entre tamaño y esfuerzos estimados. En nuestro caso, la estimación de productividad durante el mantenimiento (*EPM*) es el cociente entre las estimaciones del tamaño del cambio y el esfuerzo invertido en su realización.

$$EPM = \frac{Tamaño\ del\ cambio}{Esfuerzo\ de\ realización}$$

Una medida de la productividad interesante para tomar *a posteriori* es un índice de productividad (*IP*) que relacione el esfuerzo estimado con el esfuerzo real (empleado en el proyecto SPEM, ya referenciado en el capítulo 2).

$$IP = \frac{Esfuerzo\ estimado}{Esfuerzo\ real}$$

Considerando la dependencia que existe entre productividad en el mantenimiento y mantenibilidad, existe otro índice de productividad que denominamos I_{Mant} o índice de mantenibilidad puesto que se obtiene a partir de una medida de dicho factor de calidad. En los siguientes apartados se comenta este índice con todo detalle.

4.2.3 CRITERIO DE ASIGNACIÓN DE MANTENIBILIDAD

Como se ha indicado en el apartado anterior, la productividad es inversamente proporcional al esfuerzo, y de ahí el interés de los modelos de estimación del esfuerzo en los estudios de productividad.

El problema de la estimación del esfuerzo durante el mantenimiento es tratado en el modelo de estimación de Boehm (COCOMO). Una pieza clave en esta estimación es el **Tráfico de Cambio Anual** (TCA), que indica la proporción del programa que va a ser incluida o modificada durante un año. El esfuerzo estimado se obtiene como producto del tráfico de cambio anual y el esfuerzo de desarrollo estimado (vease apdo. 2.1.2)

$$(MM)_{MANT} = TCA\ (MM)_{DES}$$

Pero como el mismo Boehm expresa en [BOEH79] (vease apdo. 2.1.3), las características de mantenibilidad del producto van a tener una enorme influencia en el esfuerzo en

mantenimiento, características que, en nuestra opinión, no están contempladas suficientemente en los atributos directores de costos o CDA ("Cost Drivers Attributes")[8] propuestos en la versión intermedia de este modelo (vease apdo. 2.1.1). Es por ello que proponemos obtener el esfuerzo en mantenimiento como producto de tres factores: Tráfico de Cambio Anual, Esfuerzo de desarrollo y un Indice de mantenibilidad (*IMant*) que más adelante describiremos.

$$(MM)_{Mto} = TCA\ (MM)_{DES}\ \ I_{Mant}$$

El índice I_{Mant} mide los defectos de mantenibilidad, es decir, a mayores valores de I_{Mant} menor nivel de mantenibilidad.

Los valores elevados del parámetro TCA[9] provocarán un esfuerzo elevado de mantenimiento, **a no ser que,** dicha elevación del tráfico de cambios sea compensada con una reducción del valor del índice de mantenibilidad propuesto (I_{Mant}).

Otra forma de ver la relación entre el *TCA* y el I_{Mant} consiste en observar que el esfuerzo de reducir dicho índice en un módulo reporta más beneficio cuanto mayor sea el TCA. Por tanto, para maximizar el beneficio, las inversiones en mantenibilidad tendrán que ser directamente proporcionales al tráfico de cambios anual.

En consecuencia, el criterio a considerar para la asignación de características de mantenibilidad va a ser el Tráfico de Cambios Anual (TCA).

4.2.4 TRÁFICO DE CAMBIO ANUAL

Según el modelo de estimación de costos de Boehm (COCOMO, vease apdo. 2.1.2) el Tráfico de Cambio Anual es un factor que influye directamente sobre el esfuerzo en la etapa de mantenimiento,

$$TCA = \frac{NLN + NLM}{NLI}$$

[8]Los únicos atributos del software considerados son: Fiabilidad del producto requerida, Tamaño de la base de datos de la aplicación y Complejidad del producto.

[9]Téngase en cuenta que pueden ser valores superiores a la unidad, considerando la definición de TCA:
(líneas modificadas + líneas incluidas) / líneas iniciales

siendo,

NLN = Número de líneas nuevas
NLM = Número de líneas modificadas (incluidas las modificaciones sobre las incorporadas durante el mantenimiento)
NLI = Número de líneas inicial (antes de comenzar la etapa de mantenimiento)

Como se ha expresado en el apartado anterior, esta medida constituye el criterio para la asignación de características de mantenibilidad, de ahí su interés en nuestro estudio.

Es evidente que no todos los componentes de un sistema informático van a ser cambiados en igual intensidad durante la etapa de mantenimiento. En consecuencia, no es muy adecuado manejar un único valor de TCA para todo el sistema.

La Figura 10 pone de manifiesto la propuesta de clasificación en función al TCA de los componentes del software.

El aumento vertical de frecuencia de cambio va unido a un disminución de la generalización del software. Los componentes software de propósito más general requieren menos cambios que aquellos de propósito más especial y más próximos al usuario. La arquitectura multinivel de software propuesta por Denert y Thurner (véase Figura 4) sería válida para este planteamiento. En ella se muestran cuatro niveles que de mayor a menor grado de generalización, y consiguientemente de menor a mayor proximidad al usuario, son: operaciones administrativas de bases de datos, operaciones básica, operaciones de usuario y operaciones de control de diálogo.

El aumento horizontal de frecuencia, fijado un nivel de generalización, consiste un una ordenación de menor a mayor frecuencia de los **bloques funcionales** o elementos software definidos al realizar el análisis estructural en la etapa del diseño preliminar. Estos elementos se descompondrán en las siguientes etapas (diseño detallado, codificación...) en módulos concebidos para servir en el propósito común del bloque funcional. Por ejemplo, al nivel de control de diálogo, en un programa de gestión comercial, se tendrán los siguientes bloques funcionales: captura de pedidos de clientes, captura de albaranes de venta, mantenimiento del fichero o tabla de productos, etc.

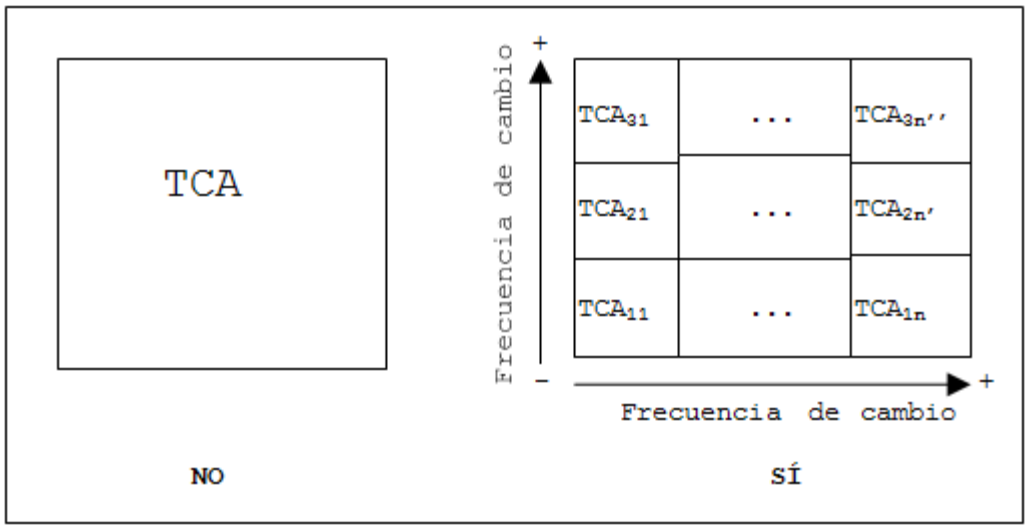


Figura 10.- Organización bidimensional de los componentes software

En lo sucesivo hablaremos indistintamente de bloques funcionales y **programas** para referirnos al mismo concepto, es decir, una parte del producto software con un propósito funcional común. Así, cada programa va a estar compuesto por módulos encargados de implementar sus funciones elementales.

Así, una de las actividades del modelo propuesto, como más tarde se verá, consiste en una estimación de la frecuencia de cambio para cada bloque funcional, es decir, la estimación de los TCA_{ij} siendo i el nivel de generalización y j el identificador del bloque funcional.

4.2.5 MÉTRICAS DE TAMAÑO Y COMPLEJIDAD DE MÓDULOS

Como es sabido, la mantenibilidad del software depende en gran medida del tamaño y la complejidad (vease [BANK93] y [STAR94] en apdo. 2.1.4).

Tamaño del módulo

Un módulo es una unidad funcional, por lo que su tamaño no puede predeterminarse. No obstante, un tamaño comúnmente aceptado varía entre 50 y 200 líneas de código, ya que tamaños superiores dificultan una visión global del mismo y su comprensión.

En caso de que, dada su complejidad, se necesiten tamaños superiores a lo indicado, se recomienda subdividir

en funciones más elementales y así construir varios módulos de tamaños adecuados.

En [CONT86] se dice al respecto: “El grado de modularidad afecta a la calidad del diseño. Tanto la sobremodularización como la inframodularización son indeseables.”

El tamaño del módulo determina en gran manera la **estructuración y modularidad** de los programas. Tamaños demasiado grandes indican una estructuración y modularidad deficiente mientras que tamaños muy pequeños indican una segmentación excesiva.

Existen dos métricas comúnmente aceptadas para medir el tamaño de un módulo:

- Líneas de Código (LDC)

- Puntos de función, métrica introducida por A. J. Albrecht [ALBR79, ALBR83], en la que se consideran varios parámetros de medida (número de entradas de usuario, número de salidas de usuario, número de peticiones al usuario, número de archivos, número de interfaces externos) que multiplicados por ciertos factores de peso y sumados dan una cuenta total. Esta cuenta total pasa por una etapa de ajuste, en base a ciertas valoraciones referentes a la complejidad, obteniendo finalmente el número de puntos de función.

La métrica considerada en nuestro modelo es la primera de ellas (**LDC**) dada su fácil extracción y su frecuente utilización en multitud de estudios.

Complejidad ciclomática

Es un concepto introducido por T. J. McCabe [MCCA76]. Consiste en una métrica del software que suministra una medida cuantitativa de la complejidad lógica de un programa, conocida como complejidad ciclomática (en adelante **CC**).

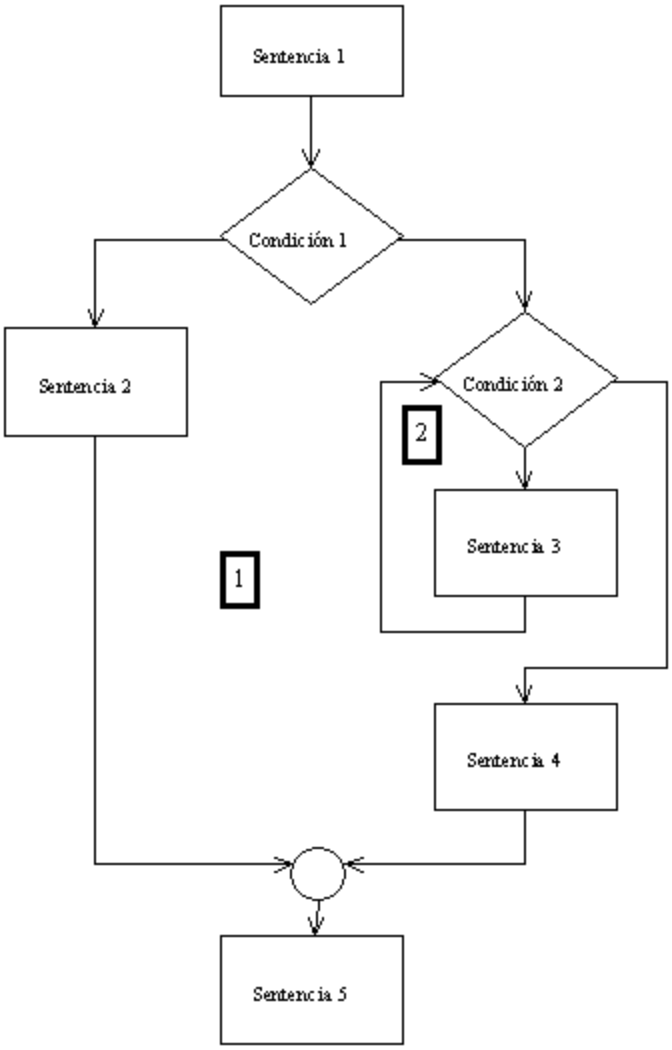
Exponemos a continuación dos modos de obtención de esta métrica:

1) A partir de un organigrama del programa.

La complejidad ciclomática CC viene dada por

$$CC = NRC + 1$$

donde NRC es el número de regiones cerradas que se observan en siguiente organigrama.



Las dos regiones cerradas están numeradas como 1 y 2 en el gráfico. Por tanto, la complejidad ciclomática será:

$$CC = NRC + 1 = 2 + 1 = 3$$

2) A partir del código fuente de un programa estructurado.

Partiendo del código fuente de un programa estructurado (en el que las instrucciones de control de flujo son: IF, WHILE, UNTIL o CASE) tenemos el siguiente método de obtención de la complejidad ciclomática:

$$CC = NIF + NWHILE + NUNTIL + \sum_{i=1}^{NCASE} (NCASOS_i - 1) + 1$$

donde:

NIF = Número de sentencias If_Then_Else
NWHILE = Número de sentencias While
NUNTIL = Número de sentencias Repeat_Until
NCASE = Número de sentencias Select/Case
$NCASOS_i$ = Número de casos para la i-ésima sentencia Select/Case

Ejemplo:

```
...
Sentencia1
If Condición1 Then
    Sentencia 2
Else
    While Condición2 Do
        Sentencia 3
    Sentencia 4
EndIf
Sentencia5
Select case Condicion3
valor1) Sentencia6
valor2) Sentencia7
valor3) Sentencia8
EndSelect
...
```

La complejidad ciclomática de este segmento de código viene dada, según el método de cálculo dado, por:

$$CC = 1 + 1 + 0 + (3 - 1) + 1 = 5$$

Comúnmente se aceptan valores de complejidad ciclomática inferiores a 10 (vease [MCCA76]). Valores superiores conllevarían una complejidad inaceptable para tareas de prueba y mantenimiento.

La complejidad ciclomática influye inversamente sobre la **sencillez** del código puesto que, a mayor complejidad, más dificil será entender un programa sin dificultad.

Así pues, atendiendo al tamaño y complejidad de los módulos, mediante el empleo de estas dos métricas, será preciso realizar una clasificación de los programas o bloques funcionales que componen el sistema en cuatro clases: TyC1, TyC2, TyC3 y TyC4 (TyC son las iniciales de Tamaño y Complejidad) (véase Figura 11).

Para ubicar un programa o bloque funcional en un grupo u otro se tomará el **tamaño medio** de todos los módulos que lo componen y la **complejidad ciclomática media**.

La necesidad de esta clasificación se observará más adelante al tratar de ajustar modelos de estimación de esfuerzo a partir de las características de mantenibilidad, ya que el ajuste a un modelo único de todos los programas sería muy difícil (con un grado de ajuste aceptable) si existe una gran diversidad en tamaño y complejidad.

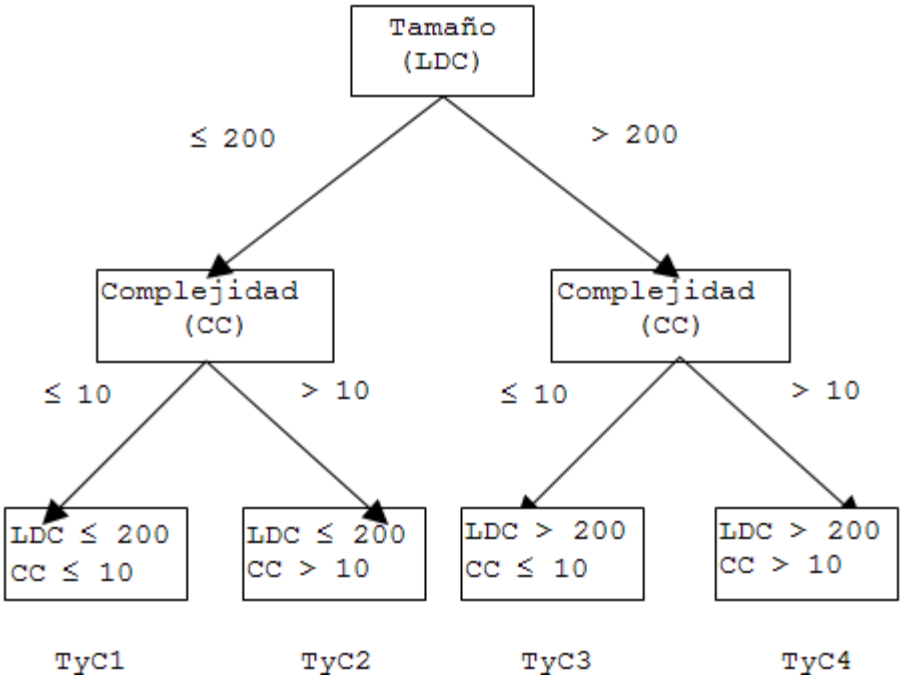


Figura 11.- Clasificación de programas según tamaño y complejidad de los módulos

La clase recomendada es la TyC1, aunque será usual encontrarse módulos en las otras clases, por lo que han de ser tenidas en cuenta en el estudio.

4.2.6 MÉTRICAS DE MANTENIBILIDAD

Las métricas de mantenibilidad propuestas a continuación han de servir para medir las características dependientes del nivel de mantenibilidad expuestas en el apartado 4.2.1, es decir, la autodocumentación, la generalización y la instrumentación. Con ello se pretende cubrir las tres etapas de realización de un cambio durante la etapa de mantenimiento: comprensión, modificación y prueba.

Métrica de comprensibilidad (X_C): **Autodocumentación**

Es el grado en que el código fuente proporciona documentación significativa. También se conoce como **documentación interna**.

La métrica escogida para medir esta característica es la proporción de líneas de comentario por cada 100 líneas de código.

La autodocumentación es, sin duda, la característica que más influye sobre la comprensibilidad del software, siendo el mejor instrumento para contender con los dos elementos causantes de la dificultad de comprensión en la etapa de mantenimiento:

a) El tiempo que dista desde el desarrollo hasta el mantenimiento. Evidentemente, con el transcurso del tiempo, hay aspectos que se olvidan, al menos en parte, con lo que la comprensión es más difícil.

b) Si el desarrollador y el mantenedor son personas distintas, es evidente que la necesidad de una buena documentación interna es mucho mayor si se pretende conseguir una fácil comprensión.

Las consecuencias de la difícil comprensión del software son lamentables al requerir grandes y agotadores esfuerzos, y pueden llegar a ser desastrosas si, por realizarse de forma precipitada, la comprensión es incorrecta.

Un aspecto a considerar en relación con la autodocumentación es que la repercusión de esta característica sobre la mantenibilidad (y por tanto sobre la productividad) depende del nivel del lenguaje. Hay lenguajes que expresan con más claridad el contenido del software que otros. Considérese un programa escrito en ensamblador (lenguaje de bajo nivel) y un programa escrito en cualquier lenguaje de alto nivel (Pascal, COBOL, etc.). Evidentemente, el programa escrito en ensamblador precisa de mucha más documentación (línea a línea) para adquirir un grado de comprensibilidad próxima a los programas escritos en lenguajes de alto nivel.

Métrica de modificabilidad (X_M): **Generalización**

Muchas de las modificaciones que han de realizarse sobre un programa consisten en:

a) Cambiar datos que están expresados de forma constante en el programa.

b) Realizar extensiones de diseño, de datos o procedimental.

La métrica propuesta consiste en medir el grado de generalización: proporción, por cada 100 líneas de código, de aquellas en las que se han extraído las constantes y las limitaciones de extensión.

Considérese un ejemplo sencillo:

Programación de un menú de selección (se escribe en un pseudocódigo).

```
FilaColumna(20,10); Pinta("1.- Primera opción");
FilaColumna(21,10); Pinta("2.- Segunda opción");
FilaColumna(22,10); Pinta("3.- Tercera opción");
FilaColumna(23,10); Pinta("Opción →");
Repite
     FilaColumna(23,20); Pide(opcion);
Hasta opcion ≥ 1   y   opcion ≤ 3
```

En este segmento de programa tenemos una gran cantidad de datos constantes. La generalización consistiría en extraer las limitaciones (datos constantes y número de items) de este código. Es decir:

- Fila y columna de la primera opción (Fila, Columna).
- Número de opciones (NOpc)
- Texto de opciones (Texto[NOpc])

El código resultante sería el siguiente:

```
Para i=1 hasta NOpc
    FilaColumna(Fila+i-1,Columna); Pinta(Texto[i]);
Fin_Para
FilaColumna(Fila+NOpc,Columna); Pinta("Opción →");
Repite
    FilaColumna(Fila+NOpc,Columna+10); Pide(opcion);
Hasta opcion ≥ 1   y   opcion ≤ NOpc
```

La generalización de esta fragmento de código implica que las modificaciones referentes a los textos de opciones, posiciones en pantalla o número de opciones se podrían realizar sin más que cambiar los parámetros indicados: Fila, Columna, NOpc, Texto[NOpc].

Métrica de testeabilidad (X_T): **Instrumentación**

Se trata de medir el grado en que el programa muestra su propio funcionamiento e identifica los errores que

aparecen. De este modo, la tareas de prueba de los cambios que se realicen van a ser mucho más fáciles.

La métrica a extraer será la proporción de líneas para el tratamiento de errores por cada 100 líneas de código.

Se pueden considerar tres tipos de errores, que deberán ser controlados en mayor o menor grado según sea al nivel de mantenibilidad requerido:

- Control de errores humanos.- Controlar que las entradas de datos concuerden con las especificaciones, de forma que el usuario no tenga la posibilidad de incluir datos incorrectos que den lugar a situaciones erróneas.

- Control de errores físicos.- Los elementos físicos, sobre los que funcionan los programas, pueden fallar. Es por ello que se requieren mecanismos de detección de tales errores y a veces incluso de corrección.

- Control de errores lógicos.- Los creadores del software también son humanos y por tanto pueden cometer fallos. El control de errores lógicos consiste en prever este tipo de fallos pudiendo ser detectados e incluso corregidos dependiendo del nivel de mantenibilidad requerido.

4.3 MODELO DE ESTIMACIÓN DEL ESFUERZO EN MANTENIMIENTO

El modelo de estimación del esfuerzo en mantenimiento (MEDEM) que proponemos, toma como punto de partida la propuesta de Boehm (en su clásico modelo COCOMO) que relaciona el esfuerzo de mantenimiento con el esfuerzo de desarrollo, donde la constante de proporcionalidad se denomina Tráfico de Cambio Anual (TCA).

$$MM_{Mto} = TCA \quad MM_{DES}$$

El esfuerzo de desarrollo se obtiene en función del tamaño del programa (KS) (vease apdo. 2.1.2). Es decir,

$$MM_{DES} = F_{DES}(KS)$$

Al considerar que el esfuerzo de mantenimiento también depende de las características de mantenibilidad del producto, se hace preciso incluir en esta ecuación un nuevo factor que constituye un **índice de la productividad de**

mantenimiento, al cual denominamos índice de mantenibilidad I_{Mant}.

$$MM_{Mto} = TCA \cdot F_{DES}(KS) \cdot I_{Mant}$$

Con TCA y KS constantes, cuanto peores sean las condiciones de mantenibilidad, mayor será el esfuerzo y por tanto mayor será el índice de mantenibilidad. Es decir, el valor del índice va a ser una medida inversa de la mantenibilidad y por tanto de la productividad en la etapa de mantenimiento.

4.3.1 DISEÑO DEL MODELO DE ESTIMACIÓN DE ESFUERZO EN MANTENIMIENTO

En primer lugar hemos de tener en cuenta que el esfuerzo en mantenimiento viene dado como la suma de sus tres componentes:

- Esfuerzo de comprensión (MM_C)
- Esfuerzo de modificación (MM_M)
- Esfuerzo de pruebas o “testing” (MM_T)

Esto motiva la descomposición del índice de mantenibilidad en tres:

- Índice de comprensibilidad (I_C)
- Índice de modificabilidad (I_M)
- Índice de testeabilidad (I_T)

Así, en lugar de tener una ecuación de estimación de esfuerzo vamos a tener tres, obteniendo la estimación global como suma de éstas:

$$MM_C = TCA \cdot F_{DES}(KS) \cdot I_C$$
$$MM_M = TCA \cdot F_{DES}(KS) \cdot I_M$$
$$MM_T = TCA \cdot F_{DES}(KS) \cdot I_T$$

Otra forma de escribirlo es:

$$MM_{Mto} = MM_C + MM_M + MM_T = TCA \cdot F_{DES}(KS) \cdot (I_C + I_M + I_T)$$

No obstante, nos interesa la primera forma ya que permite ver de forma más fácil la relación que hay entre cada componente de esfuerzo y su correspondiente índice de mantenibilidad.

Cada índice de mantenibilidad (I_C, I_M e I_T) vendrá dado por una función, que en conjunto denominamos **funciones de mantenibilidad**, donde la variable independiente es cada una de las métricas de mantenibilidad ya expuestas (X_C, X_M y X_T respectivamente) y la variable dependiente es el índice de mantenibilidad correspondiente.

$I_C = F_{IC}(X_C)$
$I_M = F_{IM}(X_M)$
$I_T = F_{IT}(X_T)$

Pasamos a ver, con más detenimiento, las formas que pueden adoptar estas funciones.

4.3.2 FUNCIONES DE MANTENIBILIDAD

Denominamos así al conjunto de las tres funciones mencionadas en el apartado anterior, que relacionan las métricas (X_C, X_M y X_T) con los índices de mantenibilidad (I_C, I_M e I_T).

El primer problema consiste en definir estas funciones, lo cual se hará partiendo de los datos históricos disponibles de proyectos anteriores. Estos datos deben estas dispuestos en tablas o en gráficos o diagramas de dispersión para ser sometidos a un análisis de regresión. El resultado será la *línea de regresión* hacia la que tienden todos los puntos del diagrama o tabla.

El método de obtención de estas funciones es idéntico para las tres, por lo que vamos a presentarlo sólo para una de ellas, por ejemplo, para la comprensibilidad.

Como ya se ha expuesto, la función F_{IC} relaciona la métrica de comprensibilidad X_C con el índice I_C.

$I_C = F_{IC}(X_C)$

Por otro lado, la relación de este índice con el esfuerzo de comprensibilidad viene dada por siguiente expresión:

$MM_C = TCA \quad F_{DES}(KS)\ I_C$

Por tanto, existe una conexión entre el esfuerzo de comprensión y la métrica de comprensibilidad, interviniendo en dicha relación dos variables más: el esfuerzo de desarrollo y la medida del tráfico de cambio anual. Así, hay

cuatro elementos de información histórica necesarios para este estudio, según se expresa en la Tabla 4:

Tabla 4.- Tabla básica de elementos de información histórica

KS	X_C	MM_C	TCA
ks_1	x_{c1}	mm_{c1}	tca_1
...	...	...	...
ks_n	x_{cn}	mm_{cn}	tca_n

KS = Bloques de mil líneas de código.
X_C = Métrica de comprensibilidad (autodocumentación).
MM_C = Esfuerzo real de comprensión (medido en la etapa de mantenimiento).
TCA= Tráfico de cambio anual.

Cada registro de esta tabla será un resumen de la actividad realizada durante un año sobre un programa o bloque funcional. Todas las medidas se tomarán anualmente, dado que todas ellas pueden ir cambiando con el tiempo.

Para cada registro de la Tabla 4 se puede obtener fácilmente el índice de mantenibilidad según la siguiente fórmula:

$$i_{Ci} = \frac{mm_{Ci}}{tca_i \quad F_{DES}(ks)}$$

Este dato se añade a la Tabla 4, dando como resultado la Tabla 5 que se muestra a continuación, con todos los elementos necesarios para el análisis de la relación que existe entre la métrica de comprensibilidad (igualmente tendríamos para la modificabilidad y la testeabilidad) y el esfuerzo de comprensión.

Tabla 5.- Tabla ampliada de elementos de información histórica

KS	X_C	MM_C	TCA	I_C
ks_1	x_{c1}	mm_{c1}	tca_1	i_{c1}
...	...	...	...	...
ks_n	x_{cn}	mm_{cn}	tca_n	i_{cn}

De esta tabla se puede extraer una subtabla con las columnas X_C e I_C, que será objeto del análisis de regresión para la obtención de la función de comprensibilidad F_{IC}.

Este es el momento de considerar la clasificación de los programas presentada en el apartado 4.2.5, ya que va a ser difícil ajustar a un único modelo toda la “nube” de puntos que vienen dados por los pares (x_{ci} , i_{ci}), en el caso

de que haya programas pertenecientes a distintas clases TyC (Tamaño y Complejidad).

Por tanto, la tabla (X_c, I_c) habrá de partirse en tantas subtablas como clases distintas de programas se den en dicho conjunto. Seguidamente, al referirnos a dicha tabla, realmente nos estaremos refiriendo a una de estas subtablas, donde todos sus elementos corresponden a programas con unas mismas características de tamaño y complejidad funcional.

4.3.2.1 ANÁLISIS DE REGRESIÓN DE LAS FUNCIONES DE MANTENIBILIDAD

Se trata de obtener una línea de regresión que se ajuste a los datos históricos disponibles (N registros) que relacionan la variable I_C con la variable X_C (extraidas de la Tabla 5).

X_C	I_C
x_{C1}	i_{C1}
...	...
x_{CN}	i_{CN}

En primer lugar habría que decidir qué tipo de línea escoger para el ajuste. Las dos formas más probables son: la forma lineal o la forma exponencial, según el crecimiento del índice en función de la métrica sea aritmético o geométrico.

Es decir, las dos funciones a probar son:

1) Función lineal

$$Y = a + b\,X$$

2) Función exponencial

$$Y = a\ e^{b\,X}$$

4.3.2.2 AJUSTE DE REGRESIÓN LINEAL

La opción más inmediata es ensayar con la forma más sencilla, es decir, con una recta o función lineal:

$$I_C^* = a + bX_C$$

donde I^*_C es el valor obtenido al aplicar la función a un valor X_C dado, que no siempre coincidirá con el valor real I_C , como es de esperar.

El método que vamos a emplear para realizar el ajuste es el conocido método de ***ajuste por mínimos cuadrados***. Se trata de obtener los parámetros de la función (en este caso *a* y *b*) que minimicen la suma de los cuadrados de los errores. Es decir, se pretende:

$$minimizar \quad \sum_{i=1}^{N}\left(I_{C_i} - I^*_{C_i}\right)^2$$

Sustituyendo en esta expresión la fórmula de obtención de I_C^*, tenemos:

$$minimizar \quad \sum_{i=1}^{N}\left(I_{C_i} - \left(a + bX_{C_i}\right)\right)^2$$

Para obtener los parámetros *a* y *b* que hacen mínima esta expresión es preciso aplicar derivadas parciales respecto de *a* y *b* igualándolas a cero.

a) $$\frac{\partial}{\partial a}\sum\left(I_{C_i} - \left(a + bX_{C_i}\right)\right)^2 = 0$$

$$\sum 2\left(I_{C_i} - \left(a + bX_{C_i}\right)\right) = 0$$

$$2\sum I_{C_i} - 2\,a\,N - 2\,b\sum X_{C_i} = 0$$

$$\boxed{\sum I_{C_i} = a\,N + b\sum X_{C_i}} \qquad \text{(Ec.1)}$$

b) $$\frac{\partial}{\partial b}\sum\left(I_{C_i} - \left(a + bX_{C_i}\right)\right)^2 = 0$$

$$\sum 2X_{C_i}\left(I_{C_i} - \left(a + bX_{C_i}\right)\right) = 0$$

$$2\sum X_{C_i}I_{C_i} - 2\,a\sum X_{C_i} - 2\,b\sum X_{C_i}^2 = 0$$

$$\boxed{\sum X_{C_i}I_{C_i} = a\sum X_{C_i} + b\sum X_{C_i}^2} \qquad \text{(Ec.2)}$$

Para mayor sencillez de las expresiones, prescindimos de anotar los límites de los sumatorios, que siempre son desde i=1 hasta N.

Tenemos pues un sistema (Ec.1 y Ec.2) de dos ecuaciones con dos incógnitas que puede ser resuelto con facilidad por cualquier método común de resolución de sistemas de ecuaciones, dando como resultado los parámetros *a* y *b* característicos de la recta de regresión.

Por tanto, para el análisis de regresión intentando un ajuste lineal, se precisan los siguiente datos acumulados (siempre para i=1 hasta N).

$\Sigma\ x_{Ci}\ i_{Ci}$	$\Sigma\ x_{Ci}$	$\Sigma\ i_{Ci}$	$\Sigma\ x_{Ci}^2$

Como todo proceso de obtención de promedios de un conjunto de valores, el cálculo de la línea de regresión debe ir acompañado de una medida de la dispersión, para conocer el grado en que el promedio obtenido puede sustituir a las observaciones de las que se obtuvo.

El ***coeficiente de determinación*** **(R^2)** es la medida que indica el grado de dispersión de la línea de regresión obtenida, y viene dado por:

$$R^2 = 1 - \frac{S_e^2}{S_y^2}$$

siendo,

S_y^2 la varianza de la variable dependiente (I_C)

S_e^2 la varianza residual correspondiente a los errores o residuos obtenidos.

$$S_y^2 = \frac{\sum y_i^2}{N} - \frac{\left(\sum y_i\right)^2}{N^2}$$

$$S_e^2 = \frac{\sum y_i^2 - a\sum y_i - b\sum x_i y_i}{N}$$

(*x* se refiere a la variable independiente (X_C) e *y* a la variable dependiente (I_C)).

El coeficiente de determinación ofrece una medida comprendida entre 0 y 1. Cuanto mayor sea, mejor será el ajuste.

4.3.2.3 AJUSTE DE REGRESIÓN EXPONENCIAL

La experiencia nos indica que lo normal no es un crecimiento lineal del esfuerzo en función de las características de mantenibilidad. Lo más frecuente es que el esfuerzo en mantenimiento siga, con respecto a los valores de mantenibilidad, una curva exponencial de exponente negativo.

Es decir, en zonas de baja mantenibilidad, un incremento unitario reporta más beneficio que en zonas de alta mantenibilidad en las que los aportes de mantenibilidad apenas suponen reducción alguna del esfuerzo, lo cual es bastante lógico. Pongamos el ejemplo de la autodocumentación. Si pasamos de no disponer de ninguna línea de comentario a unas pocas, la comprensibilidad va a mejorar considerablemente. Si ese mismo número de líneas se añaden en un módulo con una buena autodocumentación, el beneficio en la comprensibilidad va a ser bastante menor.

La forma de la función exponencial:

$$y = a\, e^{bx}$$

requiere la aplicación de logaritmos para su análisis de regresión,

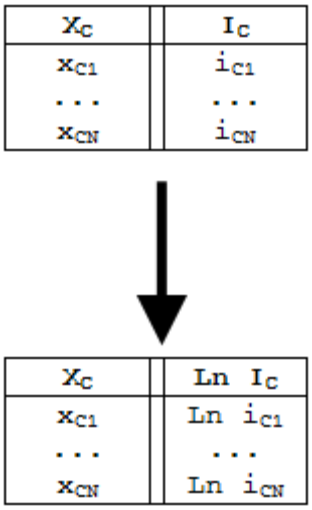
$$Ln\, y = Ln\left(a\, e^{bx}\right)$$

de donde,

$$Ln\, y = Ln\, a \; + \; bx$$

Como podemos observar, esta última ecuación corresponde a una recta, donde la variable dependiente es $Ln\ y$ y los coeficientes $Ln\ a$ y b, por lo que **se le puede aplicar el mismo método que para el ajuste lineal**.

La única diferencia radica en que hemos de cambiar los datos de entrada (aplicar logaritmos sobre el dato I_c) y que una vez obtenido el resultado hay que aplicar la función exponencial para deshacer el logaritmo.



Una vez definidas las funciones F_{IC}, F_{IM} y F_{IT}, que respectivamente transforman las métricas X_C, X_M y X_T en los índices de mantenibilidad I_C, I_M e I_T , disponemos de un modelo para estimar el esfuerzo en mantenimiento a partir de los datos ya mecionados: tráfico de cambio anual, tamaño del programa y métricas de mantenibilidad.

4.4 TÉCNICA DE ASIGNACIÓN EQUILIBRADA DE MANTENIBILIDAD

La asignación de características de mantenibilidad al producto software tiene un costo asociado. Si tales características no resultan provechosas, se está penalizando la productividad global al invertir en algo que no reporta beneficio.

Se va a proponer una técnica denominada Técnica de Asignación Equilibrada de Mantenibilidad (TAEM) que ayude a optimizar la asignación de mantenibilidad al producto de forma que se invierta en su justa medida para que el esfuerzo global sea mínimo (esfuerzo en desarrollo para mejorar la mantenibilidad más el esfuerzo de mantenimiento).

Esfuerzo de mantenimiento

Como ya se ha visto en el apartado 4.3, el esfuerzo en mantenimiento viene dado como suma de tres componentes: esfuerzo de comprensión, esfuerzo de modificación y esfuerzo de prueba.

$$MM_{Mto} = MM_C + MM_M + MM_T$$

El modelo de estimación del esfuerzo en mantenimiento propuesto (MEDEM) relaciona cada uno de los tres esfuerzos con las correspondientes métricas XC, XM y XT.

Esfuerzo de desarrollo de mantenibilidad

Para equilibrar el esfuerzo invertido en mantenibilidad con el esfuerzo de mantenimiento es preciso conocer el costo que supone las inversiones en mantenibilidad: comprensibilidad, modificabilidad y testeabilidad.

Para extraer estas medidas se necesita la siguiente información histórica:

Comprensibilidad			Modificabilidad			Testeabilidad	
Cantidad ks_{DC}	Esfuerzo mm_{DC}		Cantidad ks_{DM}	Esfuerzo mm_{DM}		Cantidad ks_{DT}	Esfuerzo mm_{DT}
ks_{DC1}	mm_{DC1}		ks_{DM1}	mm_{DM1}		ks_{DT1}	mm_{DT1}
...	...		...	...		...	...
...	...		...	...		...	...
...	...		...	...		...	...
ks_{DCn}	mm_{DCn}		ks_{DMn}	mm_{DMn}		ks_{DTn}	mm_{DTn}

Las cantidad de mantenibilidad se mide en miles de líneas (KS) que disfrutan de tal característica, y el esfuerzo en hombres-mes (MM).

- ks_{DC} Medida de miles de líneas desarrolladas para la comprensibilidad.
- ks_{DM} (Idem) para la modificabilidad.
- ks_{DT} (Idem) para la testeabilidad.
- mm_{DC} Medida del esfuerzo acumulado de desarrollo para la comprensibilidad.
- mm_{DM} (Idem) para la modificabilidad.
- mm_{DT} (Idem) para la testeabilidad.

De esta forma, para cada uno de los tres componentes de la mantenibilidad podremos obtener una medida del esfuerzo que supone la realización de mil líneas con dicha característica.

Cada registro de esta tabla acumula la cantidad y el esfuerzo correspondientes a un programa o bloque funcional. En esta tabla, como es evidente, habrá información de varios proyectos.

El esfuerzo de desarrollo medido por cada mil líneas de comprensibilidad viene dado por:

$$MM_{KSC} = \frac{\Sigma\ mm_{DC}}{\Sigma\ ks_{DC}}$$

Igualmente, los esfuerzos de desarrollo medidos correspondientes a la modificabilidad y la testeabilidad vienen dados por:

$$MM_{KSM} = \frac{\Sigma\ mm_{DM}}{\Sigma\ ks_{DM}} \qquad MM_{KST} = \frac{\Sigma\ mm_{DT}}{\Sigma\ ks_{DT}}$$

Así, tenemos los siguientes esfuerzos cuya suma, que denominamos **función objetivo,** se pretende minimizar:

Esfuerzo de desarrollo de comprensibilidad:
$MM_{DC} = MM_{KSC}\ X_C\ KS$

Esfuerzo de comprensión en mantenimiento:
$MM_C = TCA\ F_{DES}(KS)\ I_C$

La función a minimizar es:
$FO = MM_{DC} + MM_C$

Para obtener el valor mínimo, es preciso derivar la función objetivo, respecto de X_C. Al igualar dicha expresión a cero obtendremos el valor de X_C buscado.

$$FO' = MM_{KSC}\ KS + TCA\ F_{DES}(KS)\ I_C' = 0$$

Igualmente construiríamos las funciones objetivo para la modificabilidad y la testeabilidad.

4.4.1 CASO 1: FUNCIONES LINEALES DE MANTENIBILIDAD

Sea la función:
$F_{IC}(X_C) = a + b\ X_C$

Entonces la función objetivo a minimizar vendrá dada por:

$$FO = MM_{KSC}\ X_C\ KS + TCA\ F_{DES}(KS)\ (a + b\ X_C)$$

En este momento hay que realizar algunas consideraciones:

- La pendiente de la recta que determina el esfuerzo de desarrollo ha de ser positiva. No tiene sentido el

esfuerzo de una inversión sea inversamente proporcional a dicha inversión. Es decir:

$$MM_{KSC} \; X_C \; KS \geq 0$$

• La pendiente de la recta que determina el esfuerzo de mantenimiento ha de ser negativa, es decir, a mayor inversión en mantenibilidad, menor esfuerzo de mantenimiento:

$$TCA \; F_{DES}(KS) \; b \; < 0$$

Puesto que TCA y KS son necesariamente mayores que 0, tenemos que debe cumplirse:

$$b < 0$$

En este caso (función lineal) la función objetivo a minimizar debe ser analizada dentro de un rango de X_C, limitado inferior y superiormente:

$$X_C \in (0 \; , \; X_C^0)$$

donde X_C^0 es tal que $F_{IC}(X_C^0) = 0$

Obsérvese que no tiene sentido considerar valores de X_C fuera de este rango (ni negativos ni superiores a aquel que da lugar a un esfuerzo nulo de mantenimiento).

Se pueden plantear tres situaciones:

a) Que la pendiente de la recta del esfuerzo en desarrollo sea superior a la pendiente de la recta de esfuerzo en mantenimiento (en valor absoluto) (Figura 12). Dicho de otra manera, que la recta suma de ambas sea creciente o con derivada positiva:

$$MM_{KSC} \; KS \; + \; TCA \; F_{DES}(KS) \; b > 0$$

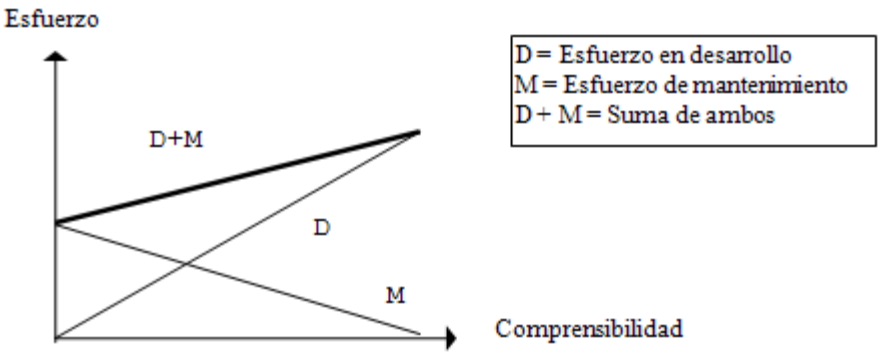


Figura 12.- Relación esfuerzo - comprensibilidad (a)

En tal caso, el mínimo dentro del intervalo fijado coincidirá con el límite inferior de dicho intervalo, es decir, $X_C = 0$.

b) Que la pendiente de la recta del esfuerzo en desarrollo sea inferior a la pendiente de la recta de esfuerzo en mantenimiento (en valor absoluto) (Figura 13). Dicho de otra manera, que la recta suma de ambas sea decreciente, es decir, que su derivada sea negativa:

$$MM_{KSC}\ KS + TCA\ F_{DES}(KS)\ b < 0$$

En tal caso, el mínimo dentro del intervalo fijado coincidirá con el límite superior de dicho intervalo, es decir, $X_C^0 \ / \ F_{IC}(X_C^0) = 0$.

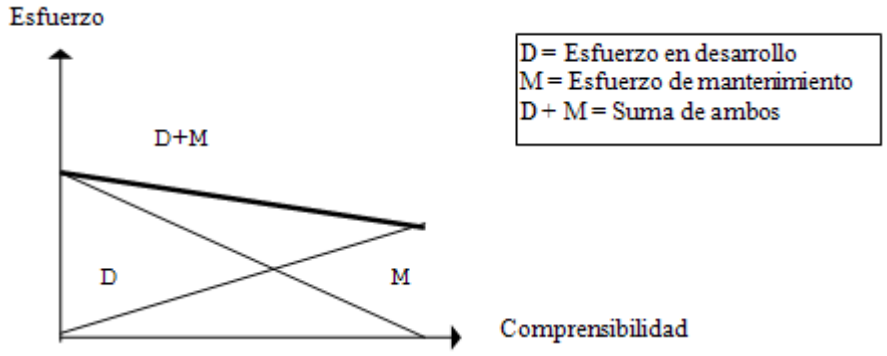


Figura 13.- Relación esfuerzo - comprensibiliad (b)

c) Que la pendiente de la recta del esfuerzo en desarrollo sea igual a la pendiente de la recta de esfuerzo en mantenimiento (en valor absoluto) (Figura 14). Dicho de otra manera, que la recta suma de ambas sea constante, es decir, que su derivada sea nula.

$$MM_{KSC}\ KS + TCA\ F_{DES}(KS)\ b = 0$$

En tal caso, cualquier valor dentro del rango dará como consecuencia el mismo resultado.

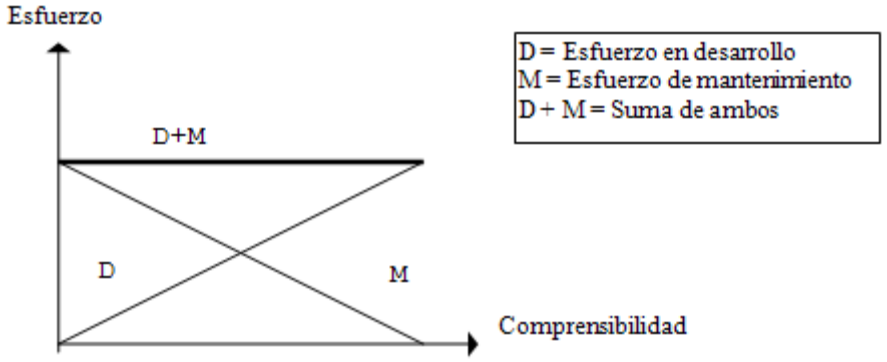


Figura 14.- Relación esfuerzo - comprensibilidad (c)

4.4.2 CASO 2: FUNCIONES EXPONENCIALES DE MANTENIBILIDAD

Sea la función:

$$F_{IC} = a\, e^{b\, X_C}$$

La función objetivo queda de la siguiente forma:

$$FO = MM_{KSC}\ \ X_C\ \ KS\ \ +\ \ TCA\ \ F_{DES}\,(KS)\ \ \left(ab^{X_C}\right)$$

En este caso, el rango de la variable independiente (X_C) tiene un límite inferior (X_C=0) ya que no se contemplan valores negativos de mantenibilidad, pero no precisa de un límite superior puesto que la función objetivo siempre va a tener un mínimo por los motivos siguientes:

- La pendiente de la recta que determina el esfuerzo de desarrollo ha de ser positiva. No tiene sentido que el esfuerzo de una inversión sea inversamente proporcional a dicha inversión. Es decir:

  $MM_{KSC}\ \ X_C\ \ KS\ \ \geq 0$

- La función exponencial que representa la mantenibilidad debe ser de exponente negativo, ya que, según aumenta la mantenibilidad decrece el esfuerzo de mantenimiento. Es decir:

  $b < 0$

Gráficamente (véase Figura 15) se puede observar que estas condiciones implican que la función objetivo, suma de la función lineal que representa el esfuerzo en desarrollo y la función exponencial que representa el esfuerzo de mantenimiento, van a tener un mínimo en el rango expresado:

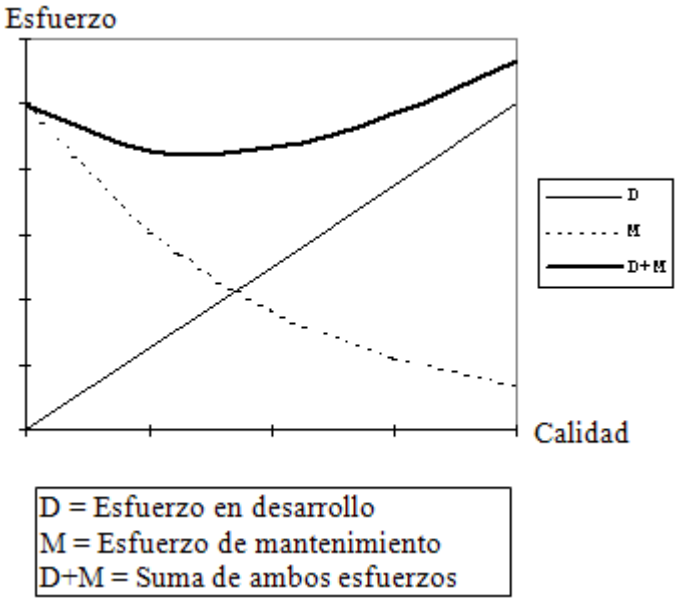


Figura 15.- Relación esfuerzo - comprensibilidad (exponencial)

Tenemos dos situaciones posibles:

a) Que la pendiente en $X_C=0$ de la función exponencial MM_C sea menor o igual que la pendiente de la recta MM_{DC} (en valor absoluto). En tal caso el mínimo dentro del rango dado será $X_C=0$ ya que el valor X_C que hace FO'=0 va a ser negativo o cero (por debajo o en el límite del rango).

b) Que la pendiente en $X_C=0$ de la función exponencial MM_C sea mayor que la pendiente de la recta MM_{DC} (en valor absoluto). En tal caso debe existir un mínimo dentro del rango que será el punto donde ambas pendientes (en valor absoluto) sean iguales. Ese punto siempre va a existir ya que la pendiente de la función exponencial tiende a 0, cuando X_C tiende a infinito (tengase en cuenta que b<0).

4.5 MODELO DE CICLO DE VIDA

Una vez visto el modelo de estimación del esfuerzo en mantenimiento, el paso siguiente será su encaje en un modelo de ciclo de vida de proyectos software, ofreciendo así una solución completa a la problemática planteada.

El modelo de ciclo de vida que se propone toma como punto de partida el presentado por J. C. Granja en [GRAN92/2], que ya se ha comentado en el apartado 2.3.2.

El motivo principal es la consideración en dicho modelo de un grupo de control de calidad con juicio de expertos, elemento que resulta necesario en nuestro modelo de productividad.

Se va a presentar, en primer lugar, los elementos o entidades que componen el modelo: Dirección del Proyecto, Equipo de Producción, Grupo de Decisión Colectiva y una herramienta de Base de Datos para almacenar la información histórica.

El siguiente paso consiste en exponer el diseño del modelo, mostrándose las interacciones entre los cuatro componentes.

Más tarde se describen los procedimientos a implementar, tomando el modelo como base, para conseguir los objetivos establecidos.

4.5.1 ELEMENTOS DEL MODELO

Este modelo está constituido fundamentalmente por cuatro elementos:

- **Dirección del proyecto (DPR)**.- Elemento activo encargado de tomar las últimas decisiones.

- **Equipo de producción (EPR)**.- Elemento activo encargado de la creación del producto, siempre sometido a las decisiones de la dirección del proyecto.

- **Grupo de decisiones colectivas (GDC)**.- Elemento activo encargado de tomar decisiones en primera instancia, haciendo uso de las opiniones de personas expertas.

- **Base de datos históricos (BDH)**.- Elemento pasivo de almacenamiento y recuperación de información histórica.

4.5.1.1 DIRECCIÓN DEL PROYECTO

En la Dirección del Proyecto (DPR) deben participar tres tipos de personas:

- Una representación de la dirección de la empresa de software que, siendo conocedores de las posibilidades económicas de la misma, tengan la capacidad de supervisar las inversiones a realizar. Estas personas tendrán la última palabra en las decisiones que impliquen consecuencias económicas.

- Una representación del equipo de producción, que ofrezca una visión precisa y actualizada de la situación del proyecto.

- Una representación del elemento operativo del grupo de decisión colectiva que sea capaz de transmitir los resultados del análisis que dicho grupo realiza a partir de las opiniones de personas expertas.

4.5.1.2 EQUIPO DE PRODUCCIÓN

Este equipo es el que realiza las tareas de producción del proyecto, acogiéndose a los estándares establecidos y a las indicaciones del GDC (siempre supervisadas por la dirección del proyecto).

Deberá facilitar las medidas del proceso solicitadas por el GDC (más concretamente, por el EDPRAI), quien dará la forma adecuada a dichas medidas para su estudio por parte del EDPI (Equipo de Decisión en Primera Instancia) o equipo de expertos.

4.5.1.3 GRUPO DE DECISIONES COLECTIVAS

Este grupo está basado íntegramente en el propuesto por J.C. Granja en [GRAN92/1, GRAN92/2], denominado Grupo de Control de Calidad (GCC) ya comentado anteriormente (véase Figura 8).

El cambio del nombre Grupo de Decisiones Colectivas en lugar de Grupo de Control de Calidad está motivado por las funciones encomendadas a dicho grupo. Su función no consiste, al menos de forma directa, en evaluar la calidad del producto, sino en realizar varios procesos de estimación basándose en las opiniones de personas expertas y anónimas, evitando así las subjetividades que surgen de forma natural

cuando las decisiones se centran en una sola persona, o en un grupo en el que es fácil que surjan elementos que obstaculicen la obtención de juicios imparciales.

4.5.1.4 BASE DE DATOS HISTÓRICOS

Para poder realizar las estimaciones, se precisa de información histórica de proyectos similares al que se encuentra en proceso de realización. En la medida en que se disponga de más o menos información histórica, y en la medida en que dicha información se asemeje al proyecto en cuestión, la estimación podrá realizarse de forma más fácil y fiable.

El contenido de la base de datos históricos (**B.D.H.**) será, a grandes rasgos, una relación entre dos conjuntos de datos (véase Figura 16).

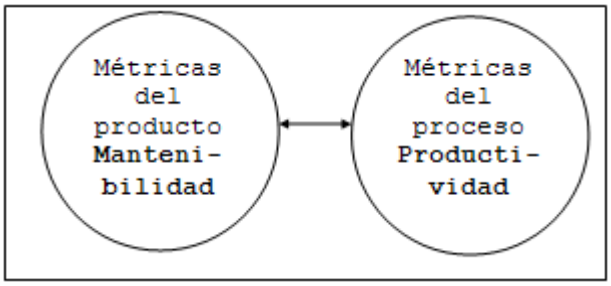


Figura 16.- Esquema simple del contenido de la B.D.H.

Es decir, se trata de relacionar medidas de la mantenibilidad del producto (vease apdo. 4.2.6) con medidas del proceso de producción. Concretamente, estas últimas son medidas del esfuerzo invertido en:

a) Concesión de características de mantenibilidad al producto.

b) Realización de cambios en la etapa de mantenimiento.

Esta información histórica, debidamente complementada con las opiniones de personas expertas que salven las diferencias entre los proyectos anteriores y el actual, será una excelente herramienta en los procesos de estimación.

4.5.2 DISEÑO DEL MODELO DE CICLO DE VIDA

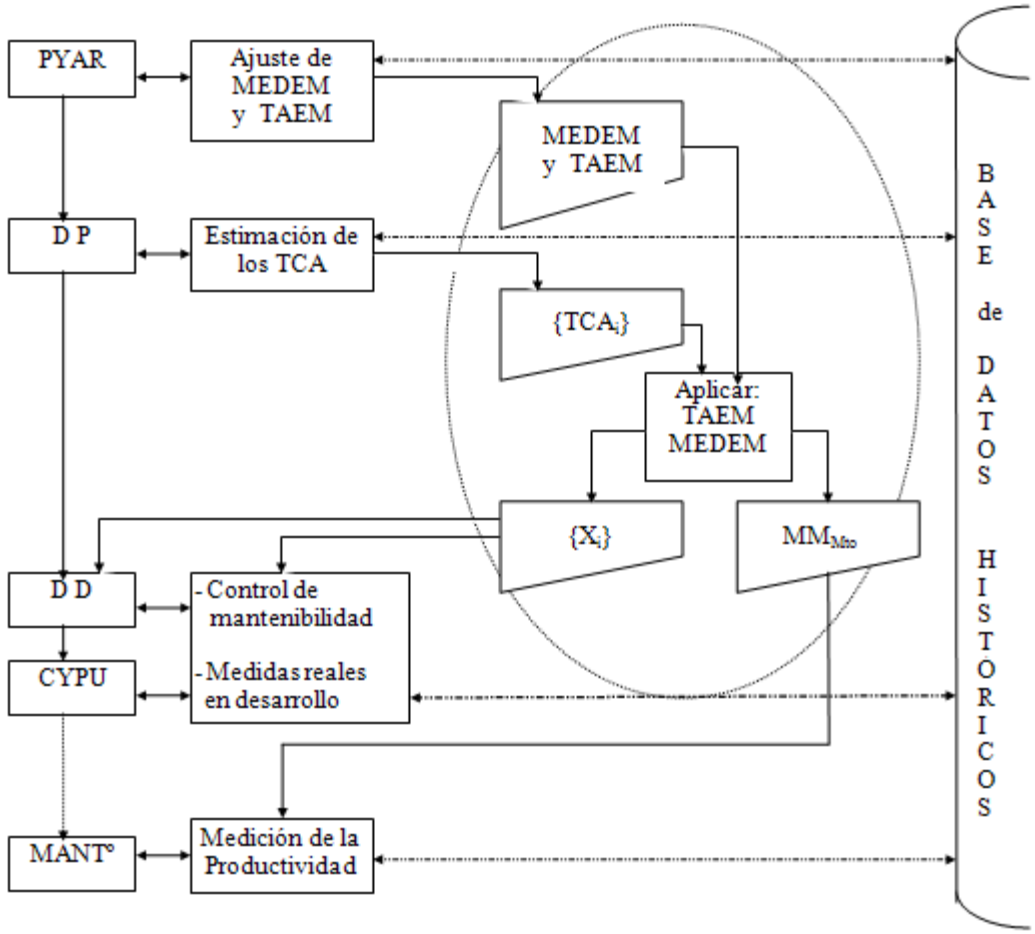
Una vez descritos los elementos del modelo, se pasará a su diseño (véase Figura 17), en el cual se observan cuatro actividades destacadas que requieren de la intervención humana:

1.- Ajuste del Modelo de Estimación Del Esfuerzo en Mantenimiento (MEDEM).

2.- Estimación del Tráfico de Cambio Anual para cada bloque funcional (TCA).

3.- Control de mantenibilidad y obtención de medidas reales en desarrollo y control de la mantenibilidad de los módulos desarrollados.

4.- Obtención de medidas reales en la etapa de mantenimiento.



PYAR.- Planificación y análisis de requisitos
DP.- Desarrollo preliminar
DD.- Desarrollo detallado
CYPU.- Codificación y prueba de unidad
MANTº.- Mantenimiento
MEDEM.- Modelo de Estimación del Esfuerzo en Mantenimiento
TAEM.- Técnica de Asignación Equilibrada de Mantenibilidad
$\{TCA_i\}$.- Tráfico de Cambio Anual para el bloque funcional *i*
$\{X_i\}$.- Valor de métricas de mantenibilidad para bloque func. i
MM_{Mto} .- Estimación del esfuerzo en mantenimiento

Figura 17.- Modelo de ciclo de vida

4.5.2.1 PROCEDIMIENTO DE AJUSTE DEL MODELO DE ESTIMACIÓN DEL ESFUERZO EN MANTENIMIENTO Y DE LA TÉCNICA DE ASIGNACIÓN EQUILIBRADA DE MANTENIBILIDAD

Al comenzar un proyecto, habrá que realizar esta actividad que consiste en ajustar los parámetros necesarios para la utilización del modelo de estimación del esfuerzo en mantenimiento (MEDEM) y de la técnica de asignación equilibrada de mantenibilidad (TAEM), a partir de los datos históricos disponibles (véase Figura 18).

Como ya se expuso en el apadrado 4.2.5, habrá que clasificar toda la información disponible en cuatro grupos correspondientes a las cuatro clases TyC_i (alguna clase podría estar vacía).

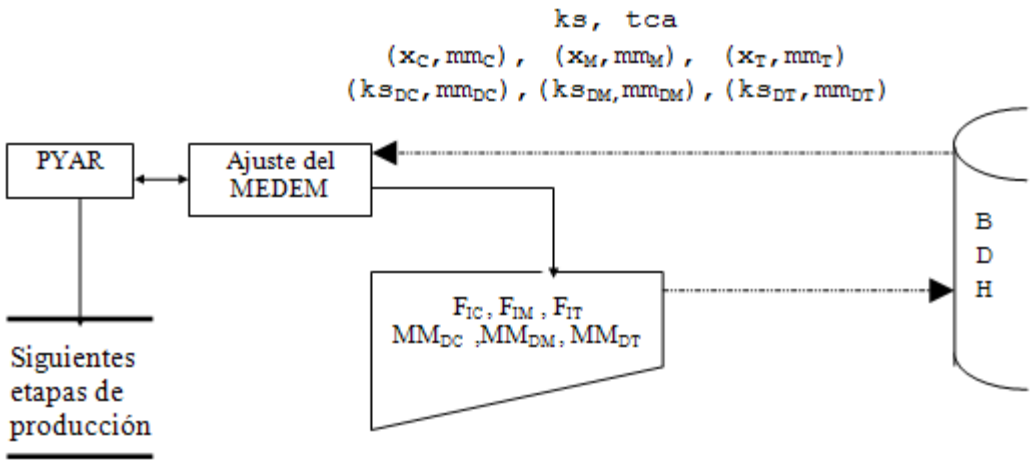


Figura 18 .- Ajuste de MEDEM y TAEM

Por cada una de las clases (que tengan elementos) habrá que hacer este mismo estudio, obteniendo probablemente un modelo distinto para cada clase.

a) OBTENCIÓN DE LAS FUNCIONES DE REGRESIÓN

La información de partida, tomada de la BDH (Base de Datos Históricos), para construir el modelo (vease apdo. 4.3.2) se observa en la Figura 19).

Cada registro o entrada de la tabla corresponderá a un resumen de la actividad realizada sobre un programa o bloque funcional. Contendrá información de diversos proyectos, con la salvedad de que todos los programas que participen en la

obtención de un modelo deben pertenecer a la misma clase TyC (Tamaño y Complejidad).

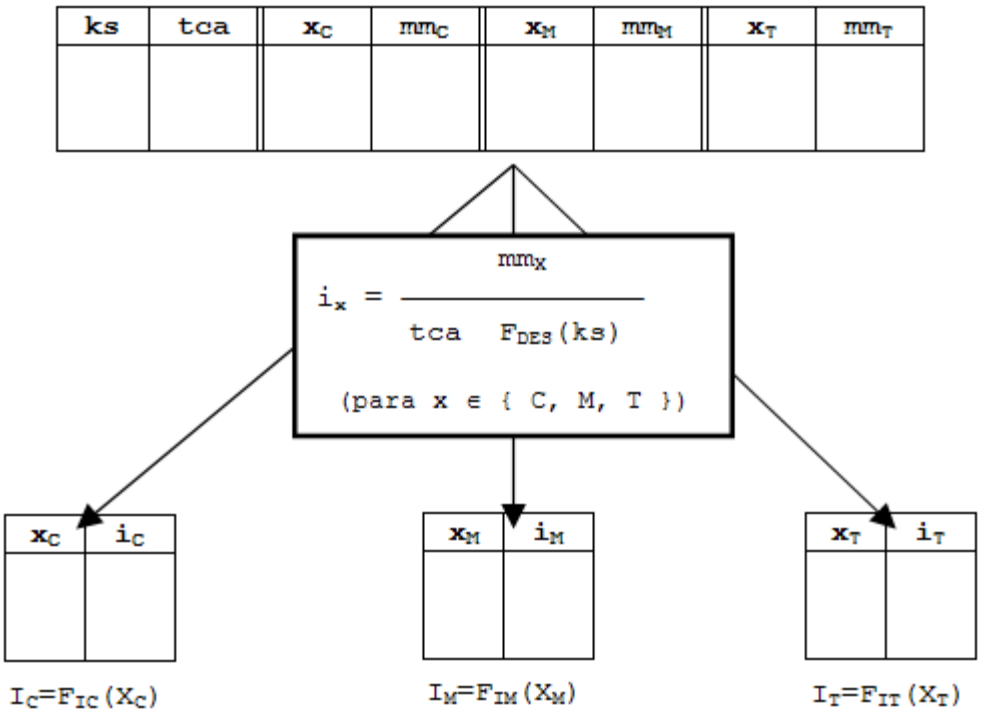


Figura 19.- Datos históricos para el ajuste de MEDEM y TAEM

Según el método expuesto en el apartado 4.3.2, se obtendrán las líneas de regresión F_{IC}, F_{IM} y F_{IT}, ajustando lineal o exponencialmente de forma que el coeficiente de determinación sea máximo.

b) OBTENCIÓN DE LOS COSTOS DE DESARROLLO DE MANTENIBILIDAD

Como ya se ha indicado en el apartado 4.4, el coste en desarrollo por cada mil líneas de código modificadas o incluidas para la mantenibilidad, se obtiene a partir de datos históricos como cociente del esfuerzo acumulado invertido en dicha tarea y el número de dichas líneas (en miles).

$$MM_{KSC} = \frac{\Sigma\ mm_{DC}}{\Sigma\ ks_{DC}} \qquad MM_{KSM} = \frac{\Sigma\ mm_{DM}}{\Sigma\ ks_{DM}} \qquad MM_{KST} = \frac{\Sigma\ mm_{DT}}{\Sigma\ ks_{DT}}$$

ks_{DC}, ks_{DM}, ks_{DT} : K-líneas modificadas o incluidas para la comprensibilidad, modificabilidad y testeabilidad, respectivamente.

mm_{DC}, mm_{DM}, mm_{DT} : Esfuerzo empleado durante el desarrollo para la comprensibilidad, modificabilidad y testeabilidad, respectivamente.

Los parámetros resultantes de esta dos actividades serán almacenadas en la BDH. Una vez se tengan los datos necesarios del proyecto en curso (estimaciones de TCA y KS para cada bloque funcional), serán empleados en la aplicación de la técnica TAEM, para obtener los valores de mantenibilidad más adecuados (X_C , X_M , X_T), así como en la aplicación del modelo MEDEM para obtener la estimación del esfuerzo en mantenimiento.

Requisitos de la B.D.H.

Esta etapa del modelo utiliza la siguiente información histórica recogida de proyectos anteriores:

Por cada proyecto:

Por cada bloque funcional o programa:

- **ks_{DC}** K-líneas para la comprensibilidad
- **mm_{DC}** Esfuerzo invertido en KS_{MDC}
- **ks_{DM}** K-líneas para la modificabilidad
- **mm_{DM}** Esfuerzo invertido en KS_{MDM}
- **ks_{DT}** K-líneas para la testeabilidad
- **mm_{DT}** Esfuerzo invertido en KS_{MDT}

Por cada año de mantenimiento:

- **tyc** Clase Tamaño y Complejidad
- **ks** Tamaño del programa
- **tca** Tráfico de cambio anual
- **x_C** Medida de la comprensibilidad
- **x_M** Medida de la modificabilidad
- **x_T** Medida de la testeabilidad
- **mm_C** Esfuerzo de comprensión
- **mm_M** Esfuerzo de modificación
- **mm_T** Esfuerzo de prueba o “testing”

4.5.2.2 ESTIMACIÓN DEL TRÁFICO DE CAMBIO ANUAL. APLICACIÓN DEL MODELO MEDEM Y LA TÉCNICA TAEM.

Tras la etapa de diseño preliminar de un proyecto, se procederá con la estimación del tráfico de cambio anual, y tras ello, la aplicación del modelo de estimación de esfuerzo en mantenimiento (MEDEM) y la técnica de asignación equilibrada de mantenibilidad (TAEM) (véase Figura 20).

El Tráfico de Cambio Anual (TCA) es un concepto empleado por Boehm en su modelo de estimación de costos (COCOMO), al que ya nos hemos referido con anterioridad. Viene dado como el cociente entre **líneas modificadas o incluidas** y **líneas iniciales**.

Para cada bloque funcional o programa, se debe realizar una estimación del TCA, que como todo proceso de estimación, está sujeto a subjetividad. Es por ello que se propone la intervención del Grupo de Decisiones Colectivas (GDC) para realizar esta estimación.

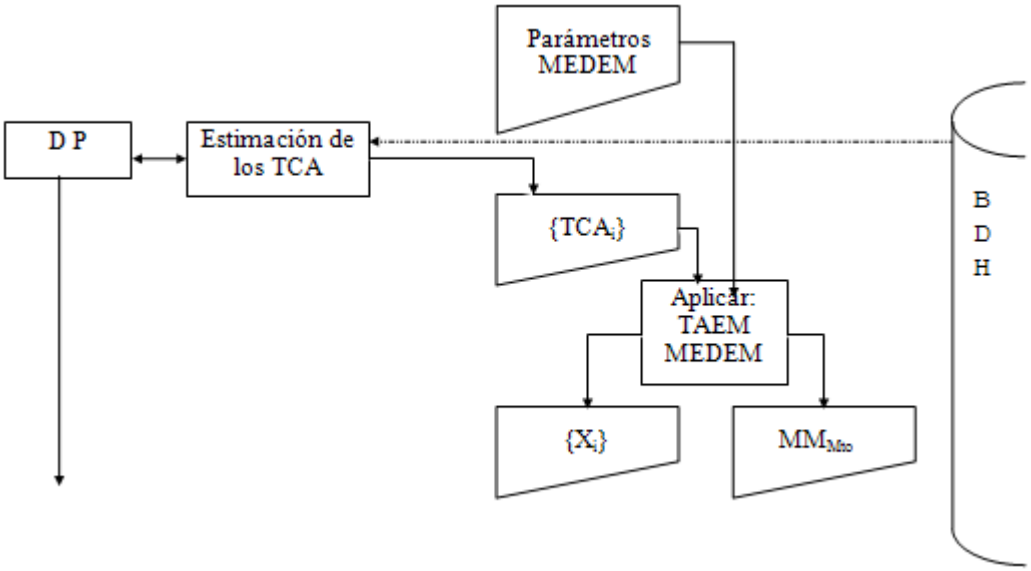


Figura 20.- Estimación de TCAs y aplicación de MEDEM y TAEM

Como ya se ha expuesto en el apartado 2.3.2, el GDC está compuesto por dos equipos humanos: EDPRAI (elemento operativo) y EDPI (elemento decisorio en primera instancia).

El procedimiento a seguir hasta la obtención de una estimación de los TCA seguirá los pasos ya indicados en el apartado 2.3.2, partiendo de la información que el EDPRAI deberá confeccionar para que el EDPI pueda realizar sus

estimaciones, y que deberá contener los siguientes elementos:

a) Breve descripción del proyecto: objetivos, amplitud, etc.

b) Por cada programa o bloque funcional se deberá especificar:
b.1) Breve descripción de la funcionalidad del programa.
b.2) Clase de frecuencia de uso (**fu**) (continua, diaria, semanal, mensual, etc.)
b.3) Relación de los agentes activos implicados, es decir, relacionados con la entrada y salidad de datos (**aai**). Algunos ejemplos de estos agentes son: personal de producción, personal administrativo, personal directivo, organismos públicos, etc.
b.4) Resumen estadístico de los datos históricos relevantes (media y desviación estándar del conjunto de datos históricos con las mismas características de este programa: frecuencia de uso y agentes activos implicados).

Una vez concluido el proceso iterativo hasta la obtención de unos resultados consensuados, se dispondrá de la estimación del TCA para cada bloque funcional: $\{TCA_i\}$.

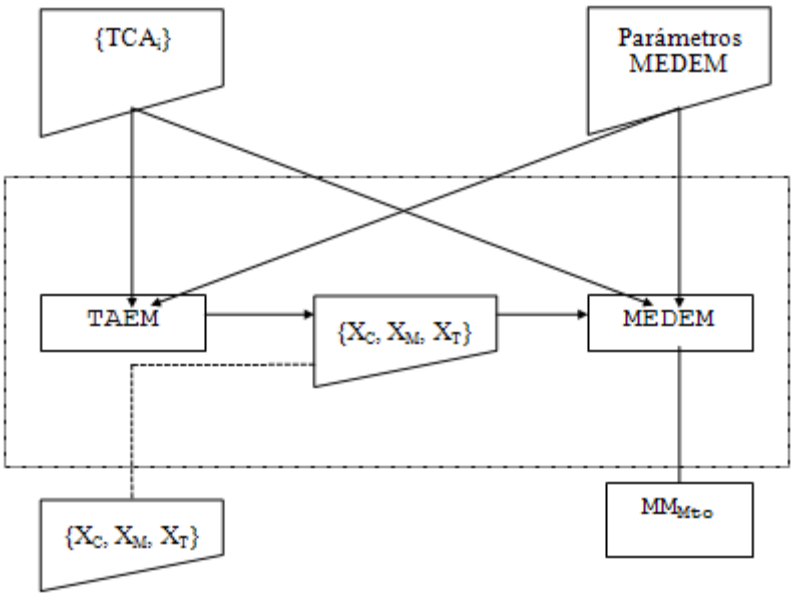


Figura 21.- Modo de aplicación de MEDEM y TAEM

En este momento se dispone de la información necesaria para aplicar la técnica de asignación equilibrada de mantenibilidad (TAEM) y la obtención del esfuerzo estimado en mantenimiento (MM_{Mto}) en hombres-mes mediante el modelo

MEDEM. En la Figura 21 se muestra el modo de aplicación de la técnica y el modelo. Como se observa, la técnica (TAEM) debe ser aplicadas antes que el modelo de estimación (MEDEM).

Requisitos de la B.D.H.

Esta etapa de estimación precisa de datos históricos procedentes de proyectos anteriores, histórica recogida durante la etapa de mantenimiento de proyectos anteriores:

Por cada proyecto:
Por cada bloque funcional o programa:
fu Frecuencia de utilización
aai Agentes activos implicados
Por cada año de mantenimiento:
tca Tráfico de cambio anual

4.5.2.3 CONTROL DE MANTENIBILIDAD Y MEDICIÓN DE ESFUERZO DE DESARROLLO DE MANTENIBILIDAD

a) CONTROL DE MANTENIBILIDAD

Durante el diseño y la codificación del producto, el equipo de producción deberá tener en cuenta las necesidades de mantenibilidad asignadas a cada bloque funcional (X_C, X_M, X_T), que han sido resultado de la anterior etapa. (Vease Figura 22).

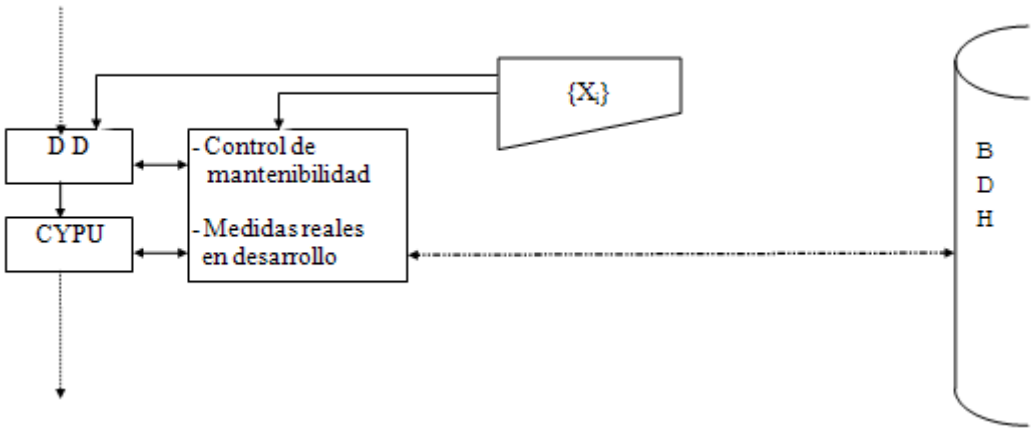


Figura 22.- Control de la mantenibilidad y medición del esfuerzo en desarrollo.

No obstante, será preciso realizar un control de tal propósito. Concluido el diseño y la codificación se deberán

extraer medidas de las características de mantenibilidad implementadas en cada módulo, comprobando que tales características satisfacen los requisitos de mantenibilidad.

b) MEDICIÓN DE ESFUERZO DE DESARROLLO DE MANTENIBILIDAD (ACTUALIZACIÓN DE B.D.H.)

Los aspectos a medir vienen dados por las necesidades de información histórica consideradas en las etapas anteriores, referentes a medidas del producto en desarrollo.

Por cada proyecto:

Por cada bloque funcional o programa:

fu Frecuencia de utilización
aai Agentes activos implicados
ks_{DC} K-líneas para la comprensibilidad
mm_{DC} Esfuerzo invertido en ks_{DC}
ks_{DM} K-líneas para la modificabilidad
mm_{DM} Esfuerzo invertido en ks_{DM}
ks_{DT} K-líneas para la testeabilidad
mm_{DT} Esfuerzo invertido en ks_{DT}

4.5.2.4 MEDICIÓN DE ESFUERZO Y PRODUCTIVIDAD EN LA ETAPA DE MANTENIMIENTO

La actividad de esta etapa está orientada a dos objetivos (véase Figura 23).

a) Grabación en la B.D.H. de medidas tomadas del proceso de realización del cambio.

b) Obtener un índice real de la productividad (el índice propuesto se obtiene como cociente del esfuerzo estimado y el esfuerzo real).

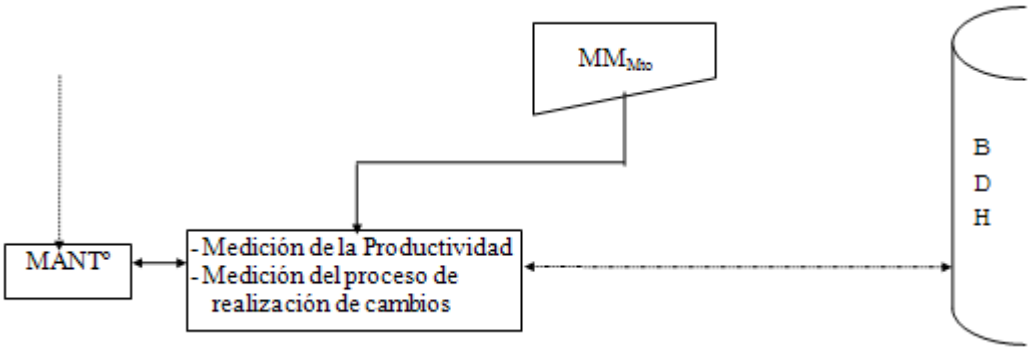


Figura 23.- Medición del esfuerzo y la productividad en la etapa de mantenimiento

a) MEDICIÓN DEL PROCESO DE REALIZACIÓN DEL CAMBIO (ACTUALIZACIÓN DE LA B.D.H.)

Los aspectos a medir vienen dados por las necesidades de información histórica consideradas en las etapas anteriores, referentes a medidas del producto en mantenimiento.

Por cada proyecto:
Por cada bloque funcional o programa:
Por cada año de mantenimiento:

ks	Tamaño del programa
tca	Tráfico de cambio anual
x_C	Medida de la comprensibilidad
x_M	Medida de la modificabilidad
x_T	Medida de la testeabilidad
mm_C	Esfuerzo de comprensión
mm_M	Esfuerzo de modificación
mm_T	Esfuerzo de prueba o “testing”

Como se observa, durante la realización de los cambios va a ser necesario diferenciar los esfuerzos dedicados a:
- la comprensión del cambio (mm_C),
- la modificación propiamente dicha (mm_M),
- las pruebas (mm_T).

b) CÁLCULO DEL ÍNDICE DE PRODUCTIVIDAD

El índice de productividad real de mantenimiento vendrá dado como cociente entre el esfuerzo estimado (resultado de la aplicación del modelo MEDEM) y el esfuerzo real ($mm_{Mto} = mm_C + mm_M + mm_T$).

$$IP = \frac{MM_{Mto}}{mm_{Mto}}$$

4.5.3 ESTRUCTURA DE LA BASE DE DATOS HISTÓRICOS

La estructura de la B.D.H. viene dada por las necesidades que impone la aplicación del modelo presentado, necesidades que han sido expuestas en cada uno de los subapartados del apartado 4.5.2.

Por tanto, la B.D.H. tendría la siguiente estructura:

Por cada proyecto:

Por cada bloque funcional o programa:

- **fu** Frecuencia de utilización
- **aai** Agentes activos implicados
- **ks_{DC}** K-líneas para la comprensibilidad
- **mm_{DC}** Esfuerzo invertido en ks_{DC}
- **ks_{DM}** K-líneas para la modificabilidad
- **mm_{DM}** Esfuerzo invertido en ks_{DM}
- **ks_{DT}** K-líneas para la testeabilidad
- **mm_{DT}** Esfuerzo invertido en ks_{DT}

Por cada año de mantenimiento:

- **tyc** Clase de tamaño y complejidad
- **ks** Tamaño del programa
- **tca** Tráfico de cambio anual
- **x_C** Medida de la comprensibilidad
- **x_M** Medida de la modificabilidad
- **x_T** Medida de la testeabilidad
- **mm_C** Esfuerzo de comprensión
- **mm_M** Esfuerzo de modificación
- **mm_T** Esfuerzo de prueba o “testing”

Para recoger estos datos, se han de construir dos tablas en la B.D.H.:

Tabla por programas

Proyecto	Programa	fu	aai	ks_{DC}	mm_{DC}	ks_{DM}	mm_{DM}	ks_{DT}	mm_{DT}

Tabla por año de mantenimiento

Proyecto	Programa	tyc	ks	tca	x_C	x_M	x_T	mm_C	mm_M	mm_T

5 APLICACIÓN DEL MODELO A VARIOS PROYECTOS SOFTWARE REALES

Tras la exposición del modelo de productividad realizada en el capítulo anterior, se plantea la demostración de la aportación de la tesis, mediante su aplicación a proyectos reales. Se han elegido tres proyectos en los que el doctorando ha participado durante su trabajo profesional para una conocida casa de software.

Se trata de tres proyectos de propósito general, desarrollados la empresa MICROJISA para usuarios de todo el territorio nacional. Los nombres de estos proyectos son: **CGFIX** (Contabilidad General y Fiscal), **GCO** (Gestión Comercial) y **LAB** (Gestión Laboral), pero en adelante y para abreviar, los llamamos **Proyecto C**¸ **Proyecto G y Proyecto A,** respectivamente. Estos proyectos han sido seleccionados de todos los que la empresa ha desarrollado por varias razones:

- Presentan los aspectos de diseño y problemática de mantenimiento y evaluación que se plantean en la tesis.
- Son proyectos en los que el doctorando ha trabajado de forma activa y directa.
- Son proyectos vivos con un gran número de instalaciones actuales y potenciales.

Estos proyectos están desarrollados, especialmente, para funcionar sobre el sistema operativo PROLOGUE[10], que posee su propio lenguaje de programación denominado BAL. Este es el lenguaje en que se han desarrollado los tres proyectos. Actualmente también se dispone de compilador de este lenguaje para UNIX, por lo que también se están explotando sobre este sistema.

La obtención de métricas comienza a realizarse en el año 1991, durante el desarrollo del proyecto G. El proyecto C se encontraba en fase de mantenimiento y el proyecto L aún no se había iniciado. Por tanto, se dispone de la siguiente información:

[10]Se trata de un sistema operativo multitarea y multipuesto, ampliamente difundido en Francia (país donde reside la empresa creadora) aunque menos conocido en nuestro país.
Es el sistema tradicionalmente empleado por la empresa MICROJISA. Actualmente no es la mejor alternativa (los nuevos desarrollos se están realizando para entornos Windows) pero, a efectos de mantenimiento, aún es el principal sistema operativo dado que la mayor parte de los clientes funcionan sobre dicho sistema.

		Mantenimiento		
Proyecto	Desarrollo	1992	1993	1994
C		X	X	X
G	X	X	X	X
L	X			X

5.1 DESCRIPCIÓN DE LA INFORMACIÓN EXTRAÍDA DE LOS PROYECTOS

Las métricas aplicadas a los proyectos son las ya descritas (vease apdo. 4.2.6), no obstante se han de hacer algunas puntualizaciones acerca del modo en que se han aplicado. También es necesario concretar la información contenida en las tablas de la B.D.H.

5.1.1 APLICACIÓN DE LAS MÉTRICAS

Se va a exponer el modo en que se han aplicado las métricas, tanto las de tamaño y complejidad como las de mantenibilidad.

5.1.1.1 MÉTRICAS DE TAMAÑO Y COMPLEJIDAD

Tamaño de módulo (LDC)

Esta medida se obtiene de forma inmediata, contando, para cada módulo, el número de líneas que lo componen.

Complejidad ciclomática (CC)

Se extrae de acuerdo a la siguiente fórmula.

$$CC = NIF + NWHILE + NUNTIL + \sum_{i=1}^{NCASE}\left(NCASOS_i - 1\right) + 1$$

NIF : Número de ocurrencias de **if**_
$NWHILE$: Número de ocurrencias de **while**_
$NUNTIL$: Número de ocurrencias de **until**_
$\sum_{i=1}^{NCASE}\left(NCASOS_i - 1\right)$: Número de ocurrencias de **case**_ menos número de ocurrencias **select**_[11]

[11]La sintaxis es:

(El símbolo _ (subrayado o guión bajo) representa aquí el espacio en blanco).

5.1.1.2 MÉTRICAS DE MANTENIBILIDAD

Estas métricas se presentan en dos formas:

Absoluta: KS_{Dx} (KS_{DC}, KS_{DM} o KS_{DT}) se refiere a miles de líneas desarrolladas para la característica de mantenibilidad correspondiente.

Porcentual: X_x (X_C, X_M o X_T) se refiere a número de líneas para la característica de mantenibilidad correspondiente por cada 100 líneas de programa.

Métrica de comprensibilidad (KS_{DC}, X_C)

Se cuentan el número de comentarios incluidos en el fuente. Concretamente, dado que el símbolo que precede a un comentario es el **';'** (punto y coma), se trata de contar las ocurrencias de este símbolo (que, evidentemente, no aparece en la sintaxis de ninguna instrucción del lenguaje).

Métrica de modificabilidad (KS_{DM}, X_M)

Se trata de obtener una medida del grado de generalización (vease apdo. 4.2.6) contando el número de líneas de código de las que se han extraído los datos constantes. Para aplicar esta métrica, se van a contar el número de líneas que no contienen datos constantes, es decir, ningún número[12] y ningún símbolo **"** (doble comilla, empleada para acotar las constantes alfanuméricas).

Métrica de testeabilidad (KS_{DT}, X_T)

```
select variable
     case valor1
          ...
     [case valor2
          ...
     ...]
     [default
          ...]
endsel
```

[12]En los proyectos estudiados, el empleo de números en los nombres de las variables es practicamente nulo.

Se trata de contar el número de líneas cuyo objetivo es el tratamiento de errores. En los estándares de trabajo de la empresa estudiada existe una variable en la que se alamacena el código de error para su posterior tratamiento. El número de líneas para tratamiento de errores va a ser directamente proporcional al número de ocurrencias de esta variable.

5.1.2 TABLAS DE LA BASE DE DATOS HISTÓRICOS

A continuación se comentan, concretando algunos aspectos necesarios para la aplicación práctica, los datos que componen las dos tablas de la B.D.H. consideradas.

Tabla por programas

fu Tipo de frecuencia de uso: Se indicará el tipo de frecuencia de uso más normal.

Con	Continua (uso continuo)
Dia	Diaria
Sem	Semanal
Men	Mensual
Tri	Trimestral
Anu	Anual
Eve	Eventual (el empleo es escaso)

aai Agentes activos implicados: Entidades que pueden tener alguna influencia sobre las necesidades del sistema.

E	Personal ejecutivo
D	Personal directivo
P	Organismos públicos (Hacienda, Seguridad Social, Consejo Superior Bancario, etc.)
O	Otras entidades (clientes, proveedores, bancos, etc.)

ks_{DC} Número de líneas desarrolladas para la comprensibilidad: Número de líneas de comentario.

mm_{DC} Esfuerzo empleado en el desarrollo de la comprensibilidad.

ks_{DM} Número de líneas desarrolladas para la modificabilidad: Número de líneas sin elementos constantes (numéricos o alfanuméricos).

mm_{DM} Esfuerzo empleado en el desarrollo de la modificabilidad.

ks_{DT} Número de líneas desarrolladas para la testeabilidad: Número de líneas dedicadas al tratamiento de errores.

mm_{DT} Esfuerzo empleado en el desarrollo de la testeabilidad.

Tabla por año de mantenimiento

tyc Clase Tamaño y Complejidad: los valores posibles son 1, 2, 3 ó 4.

1 : (TyC_1)	LDC $\leq$ 200	CC $\leq$ 10
2 : (TyC_2)	LDC $\leq$ 200	CC > 10
3 : (TyC_3)	LDC > 200	CC $\leq$ 10
4 : (TyC_4)	LDC > 200	CC > 10

(LDC = Longitud de código de un módulo
CC = Complejidad ciclomática de McCabe)

ks Tamaño del programa (en miles de líneas).

tca Tráfico de cambio anual: Proporción de código modificado o añadido durante un año.

$$TCA = \frac{NLN + NLM}{NLI}$$

NLN .- Número de líneas nuevas.
NLM .- Número de líneas modificadas.
NLI .- Número de líneas inicial (a principio de año).

x_C Medida de la comprensibilidad: Número de líneas de comentario por cada 100 líneas de programa.

x_M Medida de la modificabilidad: Número de líneas sin constantes por cada 100 líneas de programa.

x_T Medida de la testeabilidad: Número de líneas para tratamiento de errores por cada 100 líneas de programa.

mm_C Esfuerzo de comprensión: Esfuerzo total invertido en la comprensión de los cambio realizados al programa.

mm_M Esfuerzo de modificación: Esfuerzo total empleado en la realización de los cambios, propiamente dicha.

mm_T Esfuerzo de prueba o “testing”: Esfuerzo total empleado en realizar las pruebas de corrección de los cambios realizados.

5.2 DESCRIPCIÓN DE LOS PROYECTOS SOFTWARE

Proyecto C.- Contabilidad General y Fiscal. Este es un proyecto orientado a cualquier empresa, ya que se trata de un paquete de contabilidad general y fiscal, cuyas funciones son bastante evidentes: funciones contables, declaraciones fiscales, etc.

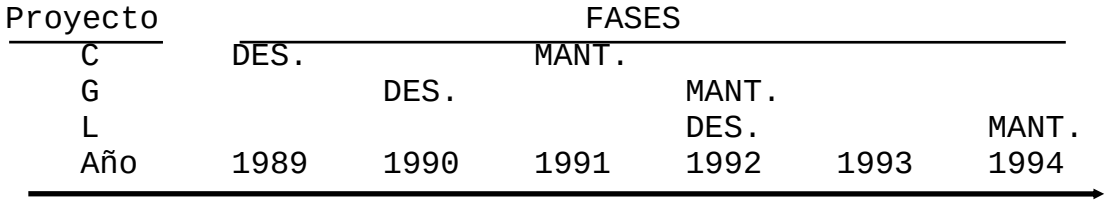
Se inició en el año 1989, comenzando su mantenimiento en el año 1991. Actualmente son más de trescientos los usuarios de esta aplicación, con unas previsiones grandes de instalaciones futuras.

Proyecto G.- Gestión Comercial. Se trata de un proyecto orientado a cualquier empresa, lo mismo que el anterior, ya que cubre sus necesidades de gestión: facturación, gestión de stock, pedidos a proveedores, pedidos de clientes, etc.

Se comenzó su desarrollo en el año 1990, comenzando su mantenimiento en el año 1992. Actualmente son unas 150 las empresas usuarias de este paquete informático, estando previsto un potencial crecimiento a corto plazo.

Proyecto L.- Gestión Laboral. Se trata de un proyecto orientado especialmente a asesorías laborales y a empresas con un número importante de trabajadores, cubriendo las necesidades laborales: contratación, elaboración de nóminas, seguros sociales, etc.

Se comenzó su desarrollo en el año 1992, comenzando su mantenimiento en el año 1994. Actualmente hay varias decenas de empresas usuarias de este paquete informático y se prevee un próximo e importante incremento.



5.3 RECOGIDA DE DATOS

Seguidamente se muestran los datos recogidos de los tres proyectos consistentes en las dos tablas ya descritas.

La columna **tyc** (grupo de tamaño y complejidad) no se incluye ya que todos los programas pertenecen al mismo grupo.

Se dispone de la siguiente información:

- Proyecto C: Tabla de mantenimiento de tres años.
- Proyecto G: Tabla de programas y tabla de mantenimiento de tres años.
- Proyecto L: Tabla de programas y tabla de mantenimiento de un año.

Las tablas de mantenimiento que se presentan son los valores medios de los años disponibles.

5.3.1 PROYECTO C

Tabla 6.- Tabla de mantenimiento (proyecto C)

Programa	ks	tca	x_C	mm_C
cgf347.s	0,293	0,62	11,60	0,086
cgfano.s	0,261	0,31	11,88	0,033
cgfapr.s	0,682	0,38	8,06	0,158
cgfbal.s	1,000	0,11	10,39	0,046
cgfbej.s	1,220	0,14	13,85	0,053
cgfbem.s	1,109	0,11	13,98	0,039
cgfcad.s	0,649	0,33	25,73	0,026
cgfcar.s	4,214	0,20	12,53	0,342
cgfcdo.s	0,464	0,18	11,42	0,033
cgfcpr.s	3,639	0,07	10,94	0,079
cgfcue.s	0,518	0,13	14,09	0,020
cgfcya.s	0,485	0,22	20,41	0,020
cgfdia.s	0,961	0,45	18,31	0,099
cgfdia1.s	0,094	0,12	7,45	0,007
cgfdio.s	0,209	0,17	20,10	0,007
cgfdpr.s	0,814	0,12	10,32	0,046
cgfeoc.s	2,216	0,23	8,71	0,289
cgfeom.s	2,075	0,15	8,87	0,178
cgfeop.s	2,584	0,14	8,63	0,211
cgfext.s	1,766	0,10	12,46	0,086
cgfinf.s	0,887	0,44	7,10	0,283
cgfinm.s	0,295	0,18	11,86	0,020
cgfins.s	0,609	0,21	14,78	0,033
cgfirp.s	0,353	0,32	17,00	0,026
cgfiva.s	0,323	0,37	17,03	0,013
cgflpr.s	0,532	0,23	15,98	0,026
cgfmoc.s	1,416	0,38	8,05	0,329
cgfmom.s	0,733	0,34	7,78	0,158
cgfout.s	0,321	0,23	9,03	0,039

cgfpfm.s	0,331	0,22	9,37	0,039
cgfrcb.s	0,532	0,16	23,31	0,007
cgfreg.s	0,427	0,10	20,37	0,007
cgfreo.s	0,576	0,24	27,78	0,007
cgfter.s	0,385	0,06	11,95	0,007
cgfuti.s	0,449	0,22	26,28	0,013

5.3.2 PROYECTO G

Tabla 7.- Tabla de programas (proyecto G)

Programa	fu	e	d	p	o	ks_{DC}	mm_{DC}
gcoage.s	eve	x				0,213	0,020
gcoalg.s	con	x				1,138	0,021
gcoart.s	eve	x				0,318	0,029
gcoavg.s	men	x				0,193	0,022
gcocar.s	men	x			x	0,127	0,021
gcocie.s	anu	x				0,378	0,067
gcocli.s	eve	x				0,091	0,045
gcocmp.s	eve	x				0,078	0,031
gcocob.s	men	x				0,211	0,015
gcocom.s	dia	x				1,090	0,005
gcocop.s	men	x				0,226	0,037
gcocos.s	men		x			0,123	0,046
gcodal.s	dia	x				0,186	0,020
gcodcp.s	eve	x				0,078	0,024
gcodic.s	eve	x				0,049	0,037
gcoegi.s	men	x				0,102	0,043
gcoema.s	eve	x				0,056	0,041
gcoenv.s	sem	x				0,417	0,026
gcoesv.s	men		x			0,268	0,033
gcoetv.s	con	x				0,033	0,051
gcoexp.s	con	x				0,186	0,038
gcofab.s	dia	x				0,582	0,049
gcofac.s	men	x				0,462	0,039
gcofam.s	eve	x				0,153	0,033
gcofas.s	eve	x				0,028	0,054
gcofcs.s	eve	x				0,035	0,029
gcoggi.s	men	x				0,091	0,044
gcogva.s	con	x				1,267	0,004
gcohes.s	men	x	x			0,153	0,066
gcohis.s	men	x	x			0,221	0,027
gcoins.s	eve	x				0,224	0,001
gcoinv.s	anu	x				0,653	0,012
gcoipg.s	eve	x				0,331	0,030

gcoitp.s	eve	x				0,269	0,006
gcolcp.s	sem	x	x			0,133	0,021
gcoldi.s	eve	x				0,157	0,065
gcolia.s	eve	x				0,151	0,095
gcoliq.s	men	x			x	0,175	0,066
gcolpc.s	sem	x				0,037	0,031
gcolve.s	men	x				0,554	0,023
gcolvi.s	dia	x				0,084	0,006
gcomgi.s	dia	x				0,095	0,071
gcomod.s	eve	x				0,020	0,119
gcompp.s	eve	x				0,117	0,069
gconiv.s	eve	x				0,026	0,056
gconov.s	eve	x				0,092	0,051
gcoofe.s	men	x				0,290	0,037
gcopac.s	eve	x				0,049	0,051
gcopag.s	eve	x				0,039	0,081
gcopal.s	eve	x				0,245	0,020
gcopar.s	eve	x				0,070	0,072
gcopav.s	eve	x				0,048	0,032
gcopcl.s	dia	x				0,758	0,041
gcopdv.s	con	x				0,161	0,009
gcoppr.s	dia	x				0,734	0,003
gcoppv.s	eve	x				0,042	0,005
gcorel.s	con	x				0,233	0,012
gcorem.s	men	x				0,087	0,035
gcosec.s	eve	x				0,024	0,001
gcosur.s	eve	x				0,034	0,003
gcotar.s	men	x			x	0,164	0,030
gcotra.s	dia	x				0,606	0,043
gcoutg.s	eve	x				0,046	0,067
gcouti.s	eve	x				0,101	0,026
gcovco.s	men	x			x	0,124	0,098
gcovel.s	men	x				0,107	0,014
gcoven.s	con	x				1,266	0,009
gcovgi.s	men	x			x	0,217	0,047
gcovis.s	dia	x				0,123	0,018
gcovve.s	men	x			x	0,282	0,079
SUMAS						17,521	2,568

Tabla 8.- Tabla de mantenimiento (proyecto G)

Programa	ks	tca	x_C	mm_C
gcoage.s	1,480	0,3	14,39	0,053
gcoalg.s	7,005	0,23	16,25	0,250
gcoart.s	4,411	0,14	7,21	0,125
gcoavg.s	1,857	0,18	10,39	0,053
gcocar.s	1,117	0,12	11,37	0,026

gcocie.s	5,053	0,28	7,48	0,316
gcocli.s	0,902	0,3	10,09	0,046
gcocmp.s	0,685	0,17	11,39	0,020
gcocob.s	3,684	0,05	5,73	0,046
gcocom.s	7,027	0,03	15,51	0,033
gcocos.s	0,833	0,27	14,77	0,039
gcocpo.s	0,218	0,13	15,6	0,007
gcodal.s	1,130	0,11	16,46	0,020
gcodcp.s	0,769	0,11	10,14	0,020
gcodic.s	0,650	0,2	7,54	0,026
gcoegi.s	0,790	0,23	12,91	0,033
gcoema.s	0,418	0,21	13,4	0,020
gcoenv.s	4,180	0,1	9,98	0,092
gcoesv.s	4,566	0,09	5,87	0,112
gcoetv.s	0,484	0,2	6,82	0,020
gcoexp.s	1,398	0,17	13,3	0,046
gcofab.s	5,294	0,21	10,99	0,237
gcofac.s	4,049	0,22	11,41	0,171
gcofam.s	1,675	0,14	9,13	0,046
gcofas.s	0,192	0,33	14,58	0,013
gcofcs.s	0,297	0,27	11,78	0,013
gcoggi.s	0,662	0,25	13,75	0,033
gcogva.s	7,966	0,11	15,91	0,099
gcohes.s	2,648	0,2	5,78	0,145
gcohis.s	3,107	0,15	7,11	0,092
gcoins.s	0,747	0,11	29,99	0,007
gcoinv.s	6,293	0,05	10,38	0,072
gcoipg.s	2,659	0,19	12,45	0,086
gcoitp.s	1,603	0,12	16,78	0,020
gcolcp.s	1,231	0,17	10,8	0,033
gcoldi.s	1,410	0,31	11,13	0,092
gcolia.s	2,367	0,27	6,38	0,178
gcoliq.s	2,122	0,23	8,25	0,112
gcolpc.s	0,311	0,14	11,9	0,007
gcolve.s	4,329	0,11	12,8	0,092
gcolvi.s	0,734	0,05	11,44	0,007
gcomgi.s	1,058	0,3	8,98	0,066
gcomod.s	0,478	0,4	4,18	0,053
gcompp.s	1,352	0,3	8,65	0,086
gconiv.s	0,305	0,27	8,52	0,020
gconov.s	1,129	0,15	8,15	0,046
gcoofe.s	2,143	0,22	13,53	0,086
gcopac.s	0,699	0,25	7,01	0,039
gcopag.s	0,508	0,24	7,68	0,033
gcopal.s	2,233	0,14	10,97	0,053
gcopar.s	1,173	0,2	5,97	0,066
gcopav.s	0,796	0,13	6,03	0,026
gcopcl.s	9,909	0,18	7,65	0,368

gcopdv.s	1,167	0,07	13,8	0,013
gcoppr.s	8,406	0,05	8,73	0,092
gcoppv.s	0,525	0,15	8	0,013
gcorel.s	2,880	0,14	8,09	0,092
gcorem.s	0,847	0,22	10,27	0,039
gcosec.s	0,359	0,21	6,69	0,020
gcosur.s	0,261	0,14	13,03	0,007
gcotar.s	3,580	0,23	4,58	0,237
gcotra.s	4,339	0,11	13,97	0,072
gcoutg.s	0,251	0,14	18,33	0,007
gcouti.s	1,176	0,17	8,59	0,046
gcovco.s	1,018	0,12	12,18	0,020
gcovel.s	1,520	0,32	7,04	0,132
gcoven.s	8,055	0,12	15,72	0,112
gcovgi.s	1,580	0,44	13,73	0,099
gcovis.s	1,171	0,33	10,5	0,092
gcovve.s	1,360	0,27	20,74	0,053

5.3.3 PROYECTO L

Tabla 9.- Tabla de programas (proyecto L)

Programa	fu	e	d	p	o	ksDC	mmDC
lab110.s	tri	x		x		0,247	0,061
lab190.s	anu	x		x		0,329	0,008
labact.s	men	x				0,225	0,038
labbor.s	eve	x				0,197	0,013
labcal.s	anu	x				0,092	0,009
labcar.s	men	x		x		0,068	0,018
labcem.s	anu	x				0,158	0,021
labcen.s	eve	x				0,052	0,043
labcer.s	anu	x				0,030	0,121
labcon.s	eve	x				0,153	0,005
labcos.s	eve	x				0,066	0,004
labcox.s	eve	x				0,066	0,003
labctr.s	men	x		x		0,391	0,047
labcve.s	anu	x		x		0,357	0,033
labepi.s	anu	x		x		0,030	0,035
labfin.s	men	x				0,056	0,043
labgal.s	eve	x				0,059	0,009
labhsi.s	eve	x				0,077	0,022
labinc.s	men	x				0,364	0,050
labins.s	eve	x				0,224	0,003
labirp.s	anu	x		x		0,068	0,048
lablno.s	men	x				0,088	0,145
lablrg.s	men	x				0,081	0,048

labnan.s	eve	x				0,084	0,020
labnom.s	men	x				2,885	0,007
labpar.s	eve	x				0,098	0,106
labpee.s	eve	x				0,123	0,040
labprd.s	eve	x				0,517	0,038
labreg.s	eve	x				0,041	0,019
labseg.s	men	x		x		1,030	0,024
labsum.s	anu	x				0,526	0,001
SUMAS						8,782	1,081

Tabla 10.- Tabla de mantenimiento (proyecto L)

Programa	ks	tca	xC	mmC
lab110.s	1,887	0,31	13,09	0,118
lab190.s	2,222	0,05	14,81	0,020
labact.s	2,075	0,27	10,84	0,092
labbor.s	1,253	0,21	15,72	0,033
labcal.s	0,643	0,13	14,31	0,013
labcar.s	0,589	0,10	11,54	0,013
labcem.s	1,456	0,11	10,85	0,026
labcen.s	0,598	0,18	8,70	0,026
labcer.s	0,154	3,50	19,48	0,053
labcon.s	1,204	0,02	12,71	0,007
labcos.s	0,275	0,14	24,00	0,007
labcox.s	0,274	0,12	24,09	0,007
labctr.s	3,409	0,23	11,47	0,158
labcve.s	3,947	0,21	9,04	0,151
labepi.s	0,580	0,13	5,17	0,020
labfin.s	0,426	0,23	13,15	0,020
labgal.s	0,257	0,15	22,96	0,007
labhsi.s	0,590	0,14	13,05	0,013
labinc.s	2,999	0,32	12,14	0,171
labins.s	0,746	0,25	30,03	0,013
labirp.s	0,643	0,21	10,58	0,026
lablno.s	0,991	0,50	8,88	0,118
lablrg.s	0,794	0,30	10,20	0,046
labnan.s	0,828	0,10	10,14	0,020
labnom.s	10,071	0,15	28,65	0,092
labpar.s	1,456	0,34	6,73	0,132
labpee.s	1,653	0,12	7,44	0,053
labprd.s	6,077	0,12	8,51	0,184
labreg.s	0,242	0,27	16,94	0,007
labseg.s	7,825	0,17	13,16	0,224
labsum.s	1,860	0,11	28,28	0,013

5.4 APLICACIÓN DEL MODELO DE ESTIMACIÓN DE ESFUERZO (MEDEM) Y DE LA TÉCNICA DE ASIGNACIÓN DE MANTENIBILIDAD (TAEM)

A continuación se procede con el análisis de los datos disponibles (Tabla 6, Tabla 7, Tabla 8, Tabla 9 y Tabla 10) según los procedimientos ya expresados.

En primer lugar se ha de realizar la etapa de ajuste, es decir, la obtención de todos los parámetros necesarios para la aplicación del modelo y la técnica.

Más tarde se mostrará la aplicación del modelo de estimación y de la técnica de asignación, una vez ajustados los parámetros que ambos incluyen.

5.4.1 AJUSTE DEL MODELO DE ESTIMACIÓN DE ESFUERZO DE DESARROLLO

Se propone un modelo lineal del tipo $Y = k\ X$, en el que la constante k se obtiene en base a los datos de los proyectos de los que se dispone información de desarrollo (G y L).

Se utiliza el conocido método de ajuste por mínimos cuadrados:

$$S = \Sigma\ (y_i - k\ x_i)^2 \quad \text{(función a minimizar)}$$

$$S' = \frac{dS}{dx} = \Sigma\ \{ -2\ x_i\ (y_i - k\ x_i) \} = 0\ ;$$

$$\Sigma\ \{ x_i\ y_i - k\ x_i^2 \} = 0\ ;$$

$$\Sigma\ x_i\ y_i - k\ \Sigma\ x_i^2 = 0\ ;$$

$$k = \frac{\Sigma\ x_i\ y_i}{\Sigma\ x_i^2}$$

	ks (x)	mm_{DES} (y)	x^2	x*y
PROYECTO C	138	48	19044	6624
PROYECTO G	229	72	52441	16488
PROYECTO L	93	24	8649	2232
sumas	460	144	80134	25344

$$k = \frac{25.344}{80.134} = 0,32$$

Así pues, la función de esfuerzo de desarrollo de acuerdo a los datos históricos disponibles es:

MM_{DES} = 0,32 KS

5.4.2 AJUSTE DE PARÁMETROS: MEDEM Y TAEM

Al inicio del proyecto es preciso ajustar aquellos parámetros necesarios para aplicar el modelo de estimación de esfuerzo en mantenimiento (MEDEM) y la técnica de asignación equilibrada de mantenibilidad (TAEM). Estos parámetros son:

Funciones de mantenibilidad

F_{IC}	Función del índice de comprensibilidad
F_{IM}	Función del índice de modificabilidad
F_{IT}	Función del índice de testeabilidad

Parámetros de esfuerzo de desarrollo

MM_{KSC}	Esfuerzo para desarrollar mil líneas para la comprensibilidad
MM_{KSM}	Esfuerzo para desarrollar mil líneas para la modificabilidad
MM_{KST}	Esfuerzo para desarrollar mil líneas para la testeabilidad

5.4.2.1 AJUSTE DE FUNCIONES DE MANTENIBILIDAD

Se trata de obtener las líneas de regresión para las funciones de mantenibilidad. Se han considerado dos modelos posibles de regresión: el modelo lineal y el exponencial. El coeficiente de determinación indicará qué modelo es más apto para cada función.

El ajuste se realiza por el método de mínimos cuadrados. Para su aplicación se necesita, por cada programa, el valor de la métrica de comprensibilidad (x_C) y el valor de índice de comprensibilidad (i_C).

La métrica de comprensibilidad x_C (vease apdo. 4.2.6) consiste en obtener el tanto por ciento de líneas de documentación interna que posee el programa, y viene dada en la tabla de mantenimiento.

El índice de comprensibilidad se obtiene a partir de la siguiente fórmula:

$$i_C = \frac{mm_C}{tca \quad 0{,}32 \; ks}$$

De acuerdo al modelo de ajuste por mínimos cuadrados, los datos que se precisan para obtener el modelo de regresión son los siguientes:

a) N (número de programas)
b) $\Sigma\ X_{Ci}$
c) $\Sigma\ I_{Ci}$
d) $\Sigma\ X_{Ci}^2$
e) $\Sigma\ I_{Ci}^2$
f) $\Sigma\ X_{Ci}\ I_{Ci}$
g) $\Sigma\ Ln(I_{Ci})$
h) $\Sigma\ (Ln(I_{Ci}))^2$
i) $\Sigma\ X_{Ci}\ Ln(I_{Ci})$

(a, b, c, d, e y f se usan en el modelo lineal)
(a, b, d, g, h e i se usan en el modelo exponencial)

Estos datos están recogidos en las tablas que se exponen a continuación (para simplificar la notación nos referiremos a x_C como **x** y a i_C como **y)**.

Proyecto C:

Programa	x (%)	y	x^2	y^2	x*y	Ly	$(Ly)^2$	x*Ly
cgf347.s	11,6	1,48	134,56	2,19	17,16	0,39	0,15	4,54
cgfano.s	11,88	1,27	141,13	1,62	15,14	0,24	0,06	2,88
cgfapr.s	8,06	1,91	64,96	3,63	15,36	0,64	0,42	5,20
cgfbal.s	10,39	1,31	107,95	1,71	13,58	0,27	0,07	2,78
cgfbej.s	13,85	0,97	191,82	0,94	13,43	-0,03	0,00	-0,43
cgfbem.s	13,98	1,00	195,44	1,00	13,97	0,00	0,00	-0,01
cgfcad.s	25,73	0,38	662,03	0,14	9,76	-0,97	0,94	-24,94
cgfcar.s	12,53	1,27	157,00	1,61	15,89	0,24	0,06	2,98
cgfcdo.s	11,42	1,23	130,42	1,52	14,10	0,21	0,04	2,41

cgfcpr.s	10,94	0,97	119,68	0,94	10,60	-0,03	0,00	-0,34
cgfcue.s	14,09	0,93	198,53	0,86	13,08	-0,07	0,01	-1,05
cgfcya.s	20,41	0,59	416,57	0,34	11,96	-0,53	0,29	-10,92
cgfdia.s	18,31	0,72	335,26	0,51	13,10	-0,33	0,11	-6,13
cgfdia1.s	7,45	1,94	55,50	3,76	14,45	0,66	0,44	4,93
cgfdio.s	20,1	0,62	404,01	0,38	12,38	-0,49	0,24	-9,75
cgfdpr.s	10,32	1,47	106,50	2,17	15,19	0,39	0,15	3,99
cgfeoc.s	8,71	1,77	75,86	3,14	15,43	0,57	0,33	4,98
cgfeom.s	8,87	1,79	78,68	3,19	15,85	0,58	0,34	5,15
cgfeop.s	8,63	1,82	74,48	3,32	15,73	0,60	0,36	5,18
cgfext.s	12,46	1,52	155,25	2,32	18,96	0,42	0,18	5,23
cgfinf.s	7,1	2,27	50,41	5,13	16,09	0,82	0,67	5,81
cgfinm.s	11,86	1,18	140,66	1,39	13,96	0,16	0,03	1,93
cgfins.s	14,78	0,81	218,45	0,65	11,92	-0,22	0,05	-3,18
cgfirp.s	17	0,72	289,00	0,52	12,23	-0,33	0,11	-5,60
cgfiva.s	17,03	0,34	290,02	0,12	5,79	-1,08	1,16	-18,38
cgflpr.s	15,98	0,66	255,36	0,44	10,61	-0,41	0,17	-6,54
cgfmoc.s	8,05	1,91	64,80	3,65	15,38	0,65	0,42	5,21
cgfmom.s	7,78	1,98	60,53	3,93	15,41	0,68	0,47	5,32
cgfout.s	9,03	1,65	81,54	2,72	14,91	0,50	0,25	4,53
cgfpfm.s	9,37	1,67	87,80	2,80	15,68	0,52	0,27	4,83
cgfrcb.s	23,31	0,26	543,36	0,07	5,99	-1,36	1,85	-31,67
cgfreg.s	20,37	0,51	414,94	0,26	10,44	-0,67	0,45	-13,62
cgfreo.s	27,78	0,16	771,73	0,03	4,40	-1,84	3,40	-51,22
cgfter.s	11,95	0,95	142,80	0,90	11,32	-0,05	0,00	-0,65
cgfuti.s	26,28	0,41	690,64	0,17	10,81	-0,89	0,79	-23,35
Sumas (35)	487,4	40,42	7907,67	58,06	460,03	-0,76	14,24	-129,91

PROYECTO G:

Programa	x	y	x^2	y^2	x*y	Ly	$(Ly)^2$	x*Ly
gcoage.s	14,39	0,37	207,07	0,14	5,37	-0,99	0,97	-14,19
gcoalg.s	16,25	0,48	264,06	0,24	7,88	-0,72	0,52	-11,76
gcoart.s	7,21	0,63	51,98	0,40	4,56	-0,46	0,21	-3,30
gcoavg.s	10,39	0,50	107,95	0,25	5,15	-0,70	0,49	-7,30
gcocar.s	11,37	0,61	129,28	0,37	6,89	-0,50	0,25	-5,69
gcocie.s	7,48	0,70	55,95	0,49	5,22	-0,36	0,13	-2,69
gcocli.s	10,09	0,53	101,81	0,28	5,36	-0,63	0,40	-6,38
gcocmp.s	11,39	0,54	129,73	0,29	6,11	-0,62	0,39	-7,09
gcocob.s	5,73	0,78	32,83	0,61	4,47	-0,25	0,06	-1,42
gcocom.s	15,51	0,49	240,56	0,24	7,59	-0,72	0,51	-11,09
gcocop.s	5,56	0,85	30,91	0,71	4,70	-0,17	0,03	-0,93
gcocos.s	14,77	0,54	218,15	0,29	8,00	-0,61	0,38	-9,05
gcodal.s	16,46	0,50	270,93	0,25	8,28	-0,69	0,47	-11,32
gcodcp.s	10,14	0,74	102,82	0,55	7,49	-0,30	0,09	-3,07
gcodic.s	7,54	0,63	56,85	0,39	4,71	-0,47	0,22	-3,54
gcoegi.s	12,91	0,57	166,67	0,32	7,33	-0,57	0,32	-7,31
gcoema.s	13,40	0,71	179,56	0,51	9,54	-0,34	0,12	-4,55
gcoenv.s	9,98	0,69	99,60	0,47	6,86	-0,37	0,14	-3,74
gcoesv.s	5,87	0,85	34,46	0,73	5,00	-0,16	0,03	-0,94
gcoetv.s	6,82	0,65	46,51	0,42	4,40	-0,44	0,19	-2,98
gcoexp.s	13,30	0,60	176,89	0,37	8,04	-0,50	0,25	-6,69
gcofab.s	10,99	0,67	120,78	0,44	7,32	-0,41	0,16	-4,46
gcofac.s	11,41	0,60	130,19	0,36	6,84	-0,51	0,26	-5,83
gcofam.s	9,13	0,61	83,36	0,38	5,60	-0,49	0,24	-4,47

gcofas.s	14,58	0,64	212,58	0,41	9,35	-0,44	0,20	-6,48
gcofcs.s	11,78	0,51	138,77	0,26	5,97	-0,68	0,46	-8,01
gcoggi.s	13,75	0,62	189,06	0,39	8,57	-0,47	0,22	-6,50
gcogva.s	15,91	0,35	253,13	0,12	5,62	-1,04	1,08	-16,56
gcohes.s	5,78	0,86	33,41	0,73	4,95	-0,16	0,02	-0,90
gcohis.s	7,11	0,62	50,55	0,38	4,39	-0,48	0,23	-3,43
gcoins.s	29,99	0,27	899,40	0,07	7,98	-1,32	1,75	-39,69
gcoinv.s	10,38	0,72	107,74	0,51	7,42	-0,34	0,11	-3,48
gcoipg.s	12,45	0,53	155,00	0,28	6,62	-0,63	0,40	-7,86
gcoitp.s	16,78	0,32	281,57	0,11	5,45	-1,12	1,26	-18,86
gcolcp.s	10,80	0,49	116,64	0,24	5,32	-0,71	0,50	-7,64
gcoldi.s	11,13	0,66	123,88	0,43	7,32	-0,42	0,18	-4,66
gcolia.s	6,38	0,87	40,70	0,76	5,55	-0,14	0,02	-0,89
gcoliq.s	8,25	0,72	68,06	0,51	5,92	-0,33	0,11	-2,74
gcolpc.s	11,90	0,50	141,61	0,25	5,98	-0,69	0,47	-8,19
gcolve.s	12,80	0,60	163,84	0,36	7,73	-0,50	0,25	-6,46
gcolvi.s	11,44	0,60	130,87	0,36	6,82	-0,52	0,27	-5,92
gcomgi.s	8,98	0,65	80,64	0,42	5,84	-0,43	0,19	-3,87
gcomod.s	4,18	0,87	17,47	0,75	3,62	-0,14	0,02	-0,60
gcompp.s	8,65	0,66	74,82	0,44	5,73	-0,41	0,17	-3,56
gconiv.s	8,52	0,76	72,59	0,58	6,47	-0,28	0,08	-2,35
gconov.s	8,15	0,85	66,42	0,72	6,92	-0,16	0,03	-1,34
gcoofe.s	13,53	0,57	183,06	0,32	7,71	-0,56	0,32	-7,60
gcopac.s	7,01	0,70	49,14	0,49	4,89	-0,36	0,13	-2,53
gcopag.s	7,68	0,85	58,98	0,72	6,50	-0,17	0,03	-1,29
gcopal.s	10,97	0,53	120,34	0,28	5,81	-0,64	0,40	-6,97
gcopar.s	5,97	0,88	35,64	0,77	5,25	-0,13	0,02	-0,77
gcopav.s	6,03	0,79	36,36	0,62	4,73	-0,24	0,06	-1,46
gcopcl.s	7,65	0,64	58,52	0,42	4,93	-0,44	0,19	-3,36
gcopdv.s	13,80	0,50	190,44	0,25	6,86	-0,70	0,49	-9,64
gcoppr.s	8,73	0,68	76,21	0,47	5,97	-0,38	0,14	-3,32
gcoppv.s	8,00	0,52	64,00	0,27	4,13	-0,66	0,44	-5,30
gcorel.s	8,09	0,71	65,45	0,51	5,77	-0,34	0,11	-2,74
gcorem.s	10,27	0,65	105,47	0,43	6,72	-0,42	0,18	-4,36
gcosec.s	6,69	0,83	44,76	0,69	5,55	-0,19	0,04	-1,25
gcosur.s	13,03	0,60	169,78	0,36	7,80	-0,51	0,26	-6,69
gcotar.s	4,58	0,90	20,98	0,81	4,12	-0,11	0,01	-0,49
gcotra.s	13,97	0,47	195,16	0,22	6,59	-0,75	0,57	-10,51
gcoutg.s	18,33	0,62	335,99	0,39	11,41	-0,47	0,22	-8,69
gcouti.s	8,59	0,72	73,79	0,52	6,18	-0,33	0,11	-2,83
gcovco.s	12,18	0,51	148,35	0,26	6,23	-0,67	0,45	-8,16
gcovel.s	7,04	0,85	49,56	0,72	5,97	-0,16	0,03	-1,16
gcoven.s	15,72	0,36	247,12	0,13	5,69	-1,02	1,03	-15,97
gcovgi.s	13,73	0,45	188,51	0,20	6,11	-0,81	0,66	-11,12
gcovis.s	10,50	0,74	110,25	0,55	7,81	-0,30	0,09	-3,11
gcovve.s	20,74	0,45	430,15	0,20	9,35	-0,80	0,63	-16,51
Sumas (70)	760,61	44,01	9515,70	29,15	444,33	-34,56	21,48	-435,61

PROYECTO L:

Programa	x	y	xx	yy	xy	Ly	Ly*Ly	x*Ly
lab110.s	13,09	0,63	171,35	0,40	8,25	-0,46	0,21	-6,04
lab190.s	14,81	0,56	219,34	0,32	8,33	-0,58	0,33	-8,52
labact.s	10,84	0,51	117,51	0,26	5,56	-0,67	0,45	-7,23
labbor.s	15,72	0,39	247,12	0,15	6,16	-0,94	0,88	-14,73

labcal.s	14,31	0,49	204,78	0,24	6,95	-0,72	0,52	-10,33
labcar.s	11,54	0,69	133,17	0,48	7,96	-0,37	0,14	-4,29
labcem.s	10,85	0,51	117,72	0,26	5,50	-0,68	0,46	-7,36
labcen.s	8,7	0,75	75,69	0,57	6,57	-0,28	0,08	-2,45
labcer.s	19,48	0,31	379,47	0,09	5,99	-1,18	1,39	-22,99
labcon.s	12,71	0,91	161,54	0,83	11,55	-0,10	0,01	-1,22
labcos.s	24	0,57	576,00	0,32	13,64	-0,57	0,32	-13,57
labcox.s	24,09	0,67	580,33	0,44	16,03	-0,41	0,17	-9,82
labctr.s	11,47	0,63	131,56	0,40	7,22	-0,46	0,21	-5,30
labcve.s	9,04	0,57	81,72	0,32	5,15	-0,56	0,32	-5,09
labepi.s	5,17	0,83	26,73	0,69	4,29	-0,19	0,04	-0,97
labfin.s	13,15	0,64	172,92	0,41	8,39	-0,45	0,20	-5,91
labgal.s	22,96	0,57	527,16	0,32	13,03	-0,57	0,32	-13,01
labhsi.s	13,05	0,49	170,30	0,24	6,42	-0,71	0,50	-9,26
labinc.s	12,14	0,56	147,38	0,31	6,76	-0,59	0,34	-7,11
labins.s	30,03	0,22	901,80	0,05	6,54	-1,52	2,32	-45,77
labirp.s	10,58	0,60	111,94	0,36	6,37	-0,51	0,26	-5,37
lablno.s	8,88	0,74	78,85	0,55	6,61	-0,30	0,09	-2,62
lablrg.s	10,2	0,60	104,04	0,36	6,16	-0,51	0,26	-5,15
labnan.s	10,14	0,75	102,82	0,57	7,65	-0,28	0,08	-2,85
labnom.s	28,65	0,19	820,82	0,04	5,45	-1,66	2,75	-47,53
labpar.s	6,73	0,83	45,29	0,69	5,61	-0,18	0,03	-1,23
labpee.s	7,44	0,83	55,35	0,70	6,21	-0,18	0,03	-1,34
labprd.s	8,51	0,79	72,42	0,62	6,71	-0,24	0,06	-2,02
labreg.s	16,94	0,33	286,96	0,11	5,67	-1,09	1,20	-18,54
labseg.s	13,16	0,53	173,19	0,28	6,93	-0,64	0,41	-8,45
labsum.s	28,28	0,20	799,76	0,04	5,62	-1,62	2,61	-45,72
Sumas (31)	446,66	17,90	7795,04	11,42	229,26	-19,19	16,99	-341,79

Ajuste de regresión lineal para la función de comprensibilidad

El ajuste consiste en obtener, a partir de la información histórica, los coeficientes de la siguiente función lineal:

$$I_C = a_C + b_C \, x_C$$

Según el método de ajuste por mínimos cuadrados, los coeficientes a_C y b_C se calculan resolviendo el siguiente sistema de ecuaciones:

$$\Sigma \, I_{Ci} = a_C \, N + b_C \, \Sigma \, X_{Ci}$$
$$\Sigma \, X_{Ci} \, I_{Ci} = a_C \, \Sigma \, X_{Ci} + b_C \, \Sigma \, X_{Ci}^2$$

Los sumatorios son desde i=1 hasta N, siendo N el número de programas disponibles en la BDH.

Según los datos mostrados en la tablas históricas anteriores, correspondientes a los tres proyectos considerados, tenemos:

$N = 136$
$\Sigma X_{Ci} = 1694,67$
$\Sigma I_{Ci} = 102,33$
$\Sigma X_{Ci}^2 = 25218,41$
$\Sigma I_{Ci}^2 = 98,63$
$\Sigma X_{Ci} I_{Ci} = 1133,62$

Así, el sistema de ecuaciones a resolver es el siguiente:

$$\begin{aligned} 136 \; a_C + 1694,67 \; b_C &= 102,33 \\ 1697,67 \; a_C + 25218,41 \; b_C &= 1133,62 \end{aligned}$$

Resultado:

$a_C = 1,18$
$b_C = -0,0345$

La bondad del ajuste se tiene con el **coeficiente de determinación (R^2)** (vease apdo. 4.3.2):

$$R^2 = 1 - \frac{S_e^2}{S_y^2}$$

$$S_y^2 = \frac{\Sigma I_{Ci}^2}{N} - \frac{(\Sigma I_{Ci})^2}{N^2}$$

$$S_e^2 = \frac{\Sigma I_{Ci}^2 - a_C \Sigma I_{Ci} - b_C \Sigma x_{Ci} I_{Ci}}{N}$$

Sustituyendo cada término por su valor ya obtenido tenemos el coefiente de determinación lineal para la comprensibilidad:

$R_{LC}^2 = 0,2254$

Ajuste de regresión exponencial para la función de comprensibilidad

La función a ajustar tiene la siguiente forma:

$$I_C = a_C \, e^{b_C x_C}$$

Al aplicar logaritmos se tiene:

$$\text{Ln } I_C = \text{Ln } a_C + b_C x_C$$

Si hacemos el cambio de variables:

$$I'_C = \text{Ln } I_C$$
$$a'_C = \text{Ln } a_C$$

tendremos la función:

$$I'_C = a'_C + b_C x_C$$

Esta es una función lineal igual a la tratada en el caso de regresión lineal, por lo que podemos aplicar el mismo método, teniendo en cuenta el cambio de variables realizado.

El sistema de ecuaciones a resolver es:

$$\Sigma I'_{Ci} = a'_C N + b_C \Sigma X_{Ci}$$
$$\Sigma X_{Ci} I'_{Ci} = a'_C \Sigma X_{Ci} + b_C \Sigma X_{Ci}^2$$

A partir de los datos históricos disponibles en las tablas de los tres proyectos estudiados, tenemos los siguientes términos (considerese que $I'_C = \text{Ln } I_C$)

$N = 136$
$\Sigma X_{Ci} = 1694,67$
$\Sigma I_{Ci} = -54,51$
$\Sigma X_{Ci}^2 = 25218,41$
$\Sigma I_{Ci}^2 = 52,70$
$\Sigma X_{Ci} I_{Ci} = -907,30$

El sistema de ecuaciones a resolver queda así:

$$136 a'_C + 1694,67 b_C = -50,51$$
$$1697,67 a'_C + 25218,41 b_C = -907,30$$

Resultado:

$$a'_C = 0,29$$
$$b_C = -0,056$$

Deshaciendo el cambio de variables ($a'_C = \text{Ln } a_C$) tenemos que:

$a_C = e^{0,29} = 1,34$

El **coeficiente de determinación** exponencial para la comprensibilidad:

$$\mathbf{R_{EC}^2 = 0,4109}$$

Puesto que $R_{EC}^2 > R_{LC}^2$ se tiene que el ajuste exponencial es mejor que el lineal.

En consecuencia, la función de comprensibilidad, escogiendo el modelo exponencial, es la siguiente:

$$I_C = F_C(X_C) = 1,34\ e^{-0,056\ Xc}$$

Siguiendo procedimientos idénticos se pueden obtener las funciones de regresión de modificabilidad y testeabilidad.

5.4.2.2 AJUSTE DE LOS PARÁMETROS DE ESFUERZO EN DESARROLLO DE MANTENIBILIDAD

Se trata de conseguir una estimación del esfuerzo para el desarrollo de la mantenibilidad. Concretamente, se mide el esfuerzo esperado para mil líneas, obteniendo valores medios partiendo de los datos históricos disponibles:

$$MM_{KSC} = \frac{\Sigma\ mm_{DC}}{\Sigma\ ks_{DC}}$$

Las funciones para modificabilidad y testeabilidad serían muy similares.

A partir de los datos históricos presentados, tenemos que el esfuerzo de desarrollo de comprensibilidad (mil líneas de comentarios) es el siguiente:

$$MM_{KSC} = \frac{3,649}{26,303} = 0,14$$

Para obtener el mejor valor de comprensibilidad para cada programa, se ha de minimizar la siguiente función objetivo:

$$FO = MM_{DC} + MM_C$$

donde,

$$MM_{DC} = MM_{KSC}\ KS\ X_C\ /\ 100$$

es el esfuerzo de desarrollo de comprensibilidad, y

$$MM_C = TCA\ MM_{DES}\ I_C$$

es el esfuerzo de comprensión durante el mantenimiento.

5.4.3 APLICACIÓN DE LA TÉCNICA DE ASIGNACIÓN EQUILIBRADA DE LA MANTENIBILIDAD (TAEM)

Una vez que se dispone del tráfico de cambio anual (TCA) para cada programa, se puede aplicar esta técnica. Su objetivo es obtener los valores más adecuados de mantenibilidad de forma que el coste total (coste de desarrollo de mantenibilidad más coste de mantenimiento) sea mínimo.

Como ya decíamos en el apartado anterior, el esfuerzo de desarrollo de comprensibilidad es,

$$MM_{DC} = MM_{KSC} \; KS \; X_C \; / \; 100$$

Igualmente tenemos los esfuerzos de desarrollo de modificabilidad y testeabilidad:

$$MM_{DM} = MM_{KSM} \; KS \; X_M \; / \; 100$$
$$MM_{DT} = MM_{KST} \; KS \; X_T \; / \; 100$$

OBTENCIÓN DEL VALOR DE COMPRENSIBILIDAD (X_C)

Tenemos los siguientes valores de esfuerzo:
a) El esfuerzo de desarrollo de comprensibilidad:
$MM_{DC} = MM_{KSC} \; KS \; X_C \; / \; 100$

b) Esfuerzo de comprensión en mantenimiento:
$MM_C = TCA \; MM_{DES} \; a_c \; e^{bc \; Xc}$

La función a minimizar es:
$FO = MM_{DC} + MM_C$

Al sustituir las expresiones de MM_{DC} y MM_C queda:
$FO = MM_{KSC} \; KS \; X_C \; /100 \; + \; TCA \; MM_{DES} \; a_c \; e^{bc \; Xc}$

Derivando respecto de X_C tenemos:

$$FO' = MM_{KSC} \; KS \; / \; 100 + \; TCA \; MM_{DES} \; a_c \; b_c \; e^{bc \; Xc}$$

Igualando a cero:

$$MM_{KSC} \; KS \; / \; 100 + \; TCA \; MM_{DES} \; a_c \; b_c \; e^{bc \; Xc} = 0$$

Despejando X_C queda:

$$X_C = \frac{\text{Ln } (MM_{KSC}\ KS\ /\ 100) - \text{Ln } (-\ TCA\ \ MM_{DES}\ a_c\ b_c)}{b_C}$$

(Téngase en cuenta que b_C ha de ser menor que 0 (vease apdo. 4.3.2). De ahí el sentido del signo negativo dentro del segundo logaritmo del numerador).

Como se puede observar, esta es una función que relaciona el tráfico de cambio anual con el valor de comprensibilidad óptimo.

Sustituyendo las variables por sus valores tenemos,

$$X_c = \frac{\text{Ln}\left(\frac{0{,}14\ KS}{100}\right) - \text{Ln}\left(-\text{TCA}\ \ 0{,}32\ KS^{\ 1{,}34}\ (-0{,}056)\right)}{-0{,}056}$$

y por tanto,

$$X_C = \frac{2{,}84 + \text{Ln TCA}}{0{,}056}$$

Ejemplo:

TCA	X_C
0,10	9
0,25	26
0,50	38
0,75	46

Como se deduce fácilmente (igualando a cero el numerador de la expresión anterior), el valor mínimo de TCA para que el valor X_C recomendado sea positivo es:

$$TCA_{MIN} = e^{-2.84} = 0{,}06$$

Es decir, si el tráfico de cambio anual de un programa durante el mantenimiento está por debajo del 6% entonces no resulta productivo incidir en la comprensibilidad del programa (incluyendo líneas de comentario).

De idéntica manera se obtendrían las funciones de cálculo de los valores de modificabilidad y testeabilidad óptimos.

5.4.4 APLICACIÓN DEL MODELO DE ESTIMACIÓN DEL ESFUERZO DE MANTENIMIENTO (MEDEM)

Una vez conocidos, para cada programa, el tráfico de cambio anual (TCA) y los valores de mantenibilidad (X_C, X_M y X_T) se procede a la aplicación del modelo de estimación del esfuerzo en mantenimiento:

$$MM_{Mto} = MM_C + MM_M + MM_T$$

donde,

$$MM_C = TCA \quad MM_{DES} \quad a_c \; e^{bC \; xC}$$
$$MM_M = TCA \quad MM_{DES} \quad a_M \; e^{bM \; xM}$$
$$MM_T = TCA \quad MM_{DES} \quad a_T \; e^{bT \; xT}$$

Esta medida (MM_{Mto}) indica el esfuerzo esperado de mantenimiento durante un año (ya que TCA es el tráfico de cambio anual).

Como venimos haciendo en la exposición de este caso práctico, vamos a centrarnos en el esfuerzo de comprensión, es decir,

$$MM_C = TCA \quad MM_{DES} \quad a_c \; e^{bC \; xC}$$

Aplicando los datos disponibles tenemos la siguiente función de cálculo del esfuerzo de comprensión:

$$MM_C = TCA \; 0{,}32 \; KS \; 1{,}34 \; e^{-0{,}056 \; x_c}$$

Un caso particular se da cuando el valor de X_C coincide con el recomendado por la técnica de asignación (TAEM). Entonces tenemos que el esfuerzo es directamente proporcional al tamaño del programa y no depende del TCA.

Sustituyendo en la expresión anterior X_C por el valor que la técnica TAEM recomienda, tenemos:

$$MM_C = TCA \; 0{,}32 \; KS \; 1{,}34 \; e^{-0{,}056 \; \frac{2{,}84 + Ln \; TCA}{0.056}}$$

Y simplificando esta expresión se obtiene una relación de proporcionalidad entre KS y MM_C,

$$MM_C = 0{,}025 \quad KS$$

6 OTRAS APORTACIONES AL MODELO DE PRODUCTIVIDAD

La principal aportación que hacemos al modelo en este capítulo consiste en describir su integración en los entornos de programación, que es uno de los objetivos de la tesis. Se incluyen otras dos pequeñas aportaciones a este campo que aún deben ser desarrolladas con más profundidad, por lo que se referencian en el apartado de líneas de trabajo futuro.

6.1 INTEGRACIÓN DEL MODELO EN UN ENTORNO DE PROGRAMACIÓN

En este apartado se realiza una propuesta de integración del modelo de productividad en los entornos de programación. Se comienza exponiendo cuáles son las responsabilidades encomendadas al entorno, es decir, qué aspectos son total o parcialmente automatizables. Seguidamente se describen las herramientas necesarias para satisfacer tales requisitos. Por último se concreta el modo en que estas herramientas se integran en los entornos de programación.

6.1.1 ACTIVIDADES AUTOMATIZABLES DEL MODELO DE PRODUCTIVIDAD

La aceptación en un proyecto software del modelo y técnica propuestos va a depender en gran medida de la facilidad de integración de éstos en los entornos de programación.

Las siguientes actividades pueden ser incorporadas en herramientas:

a) Extracción de métricas del producto.
b) Extracción de métricas del proceso de producción.
c) Ajuste de parámetros (MEDEM y TAEM).
d) Aplicación de la técnica de asignación equilibrada de mantenibilidad (TAEM).
e) Aplicación del modelo de estimación de esfuerzo de mantenimiento (MEDEM).
f) Gestión de la base de datos históricos (BDH).

Veamos pues, qué aspectos del modelo son totalmente automatizables y cuáles requieren la dirección humana.

a) Extracción de métricas del producto.

Considerando las métricas propuestas, ésta es una actividad totalmente automatizable. Así, una simple herramienta podrá obtener para cada programa estas tres métricas:

- Número de líneas de comentario.
- Número de líneas sin constantes.
- Número de líneas para el tratamiento de errores.

b) Extracción de métricas del proceso de producción.

Como se ha visto en la exposición del modelo, se necesita tomar médidas de esfuerzo tanto en desarrollo (en la etapa de codificación) como en mantenimiento. Durante la etapa de codificación de un programa es preciso distinguir entre el esfuerzo de desarrollo de la funcionalidad y el de desarrollo de las características de mantenibilidad. En nuestro modelo, este último esfuerzo se refiere a las siguientes tareas:

- Inclusión de comentarios.
- Generalización (sustitución de constantes por variables).
- Tratamiento de errores.

Durante el mantenimiento se deben distinguir los esfuerzo dedicados a:

- La comprensión del cambio a realizar.
- La modificación propiamente dicha.
- La prueba de corrección del cambio.

En el proceso de extracción de estas medidas debe eliminarse la intevención humana siempre que sea posible, facilitando así la aceptación del modelo por parte del equipo de producción.

De no facilitar dicha integración, la actividad de control de niveles de mantenibilidad (vease apdo. 4.5.2) puede suponer un impedimento para que el equipo de producción acepte dicho modelo de productividad, dada la necesidad continua de extraer métricas del producto con objeto de validar su nivel de mantenibilidad.

c) Ajuste de parámetros (MEDEM y TAEM).

El ajuste de los parámetros necesarios para la aplicación del modelo de estimación del esfuerzo en mantenimiento (MEDEM) y de la técnica de asignación equilibrada de mantenibilidad (TAEM), está perfectamente determinado (apartados 4.3 y 4.4) y se obtiene de forma automática a partir de los datos históricos disponibles.

d) Aplicación de la técnica de asignación equilibrada de mantenibilidad (TAEM).

Una vez determinados los bloques funcionales o programas a desarrollar y obtenida la estimación del tráfico de cambio anual (TCA) para cada programa, se puede proceder a la aplicación de la técnica de asignación equilibrada de mantenibilidad. Esta técnica, previamente ajustada, ofrece automáticamente los valores de mantenibilidad que resultan más adecuados en función del TCA.

e) Aplicación del modelo de estimación de esfuerzo de mantenimiento (MEDEM).

Al igual que la técnica TAEM, el modelo de estimación de esfuerzo de mantenimiento funciona de forma totalmente automática. Dado el tráfico de cambio anual (TCA) para cada programa, así como los valores de mantenibilidad (comprensibilidad, modificabilidad y testeabilidad), el modelo proporciona una estimación del esfuerzo de mantenimiento.

f) Gestión de la base de datos históricos (BDH).

La gestión de la base de datos históricos, es también una actividad en gran parte automatizable (almacenamiento de tablas históricas, obtención de informes, extracción de información estadística, etc.). Igualmente, el entorno ofrecerá un soporte para mantener los documentos que intecambian los distintos elementos del modelo (dirección del proyecto, equipo de producción y grupo de decisiones colectivas).

6.1.2 HERRAMIENTAS DE CONTROL DE LA PRODUCTIVIDAD

A continuación se describen las características que deben poseer las herramientas destinadas a cubrir las necesidades expuestas en el apartado anterior. A grandes rasgos, se necesita disponer de herramientas de tres tipos:

a) Herramientas de medición.
b) Herramientas de administración de datos.
c) Herramientas de cálculo.

1) Herramienta de extracción de métricas del producto.

La obtención de las métricas consideradas es bastante fácil y puede llevarse a cabo mediante un sencillo **analizador de programas**:

a) Comprensibilidad.- La métrica de comprensibilidad viene dada por el número de líneas de comentarios. El programa analizador reconocerá el comienzo y fin de un comentario contabilizando las líneas contenidas.

b) Modificabilidad.- Esta métrica viene dada por el número de líneas sin datos constantes. Su obtención es tan sencilla como contabilizar aquellas líneas que no contienen ninguna constate numérica ni alfanumérica (normalmente encerrada entre comillas).

c) Testeabilidad.- Esta métrica viene dada por el número de líneas dedicadas al tratamiento de errores. Las instrucciones de manejo de errores suelen tener características distintivas del resto de las instrucciones.

2) Editor con control de tiempos.

El editor (o editores) a emplear contituye uno de los principales elementos para el éxito en la implementación de este modelo de productividad. Esta herramienta debe facilitar la tarea de medir los tiempos de edición de acuerdo a la siguiente clasificación:

a) Tiempo de desarrollo de la funcionalidad.
b) Tiempo de asignación de mantenibilidad.
    b1) Tiempo dedicado a la comprensibilidad.
    b2) Tiempo dedicado a la modificabilidad.
    b3) Tiempo dedicado a la testeabilidad.
c) Tiempo de mantenimiento.
    c1) Tiempo de comprensión.
    c2) Tiempo de modificación.
    c3) Tiempo de pruebas.

Es fácil distinguir entre el esfuerzo de codificación de las características funcionales del programa[13], el esfuerzo de asignación de características de mantenibilidad[14], y el esfuerzo realizado durante el mantenimiento. Así, este primer nivel de clasificación se encomienda al codificador. Es decir, el entorno facilitará una combinación de teclas (o un “botón”, en entornos de ventanas) que permita conmutar entre estos tipos de actividades.

[13]El codificador o programador tiene en mente las necesidades funcionales asignadas al programa.
[14]El programador o codificador tiene en mente las necesidades de mantenimiento (facilidad de comprensión, de modificación y de localización de errores).

El control manual del segundo nivel de clasificación para el tiempo de mantenibilidad puede resultar bastante engorroso para el programador. Un **editor de estructuras** simplifica bastante esta labor, eliminando la necesidad de la intervención humana, ya que el editor "sabe", en todo momento, qué estructura se encuentra en edición. Para nuestro propósito interesa conocer si la estructura en edición es de comentario (comprensibilidad) o de manejo de errores (testeabilidad). Por exclusión, el tiempo dedicado a la mantenibilidad que no corresponde a comprensibilidad ni testeabilidad se entiende dedicado a la modificabilidad.

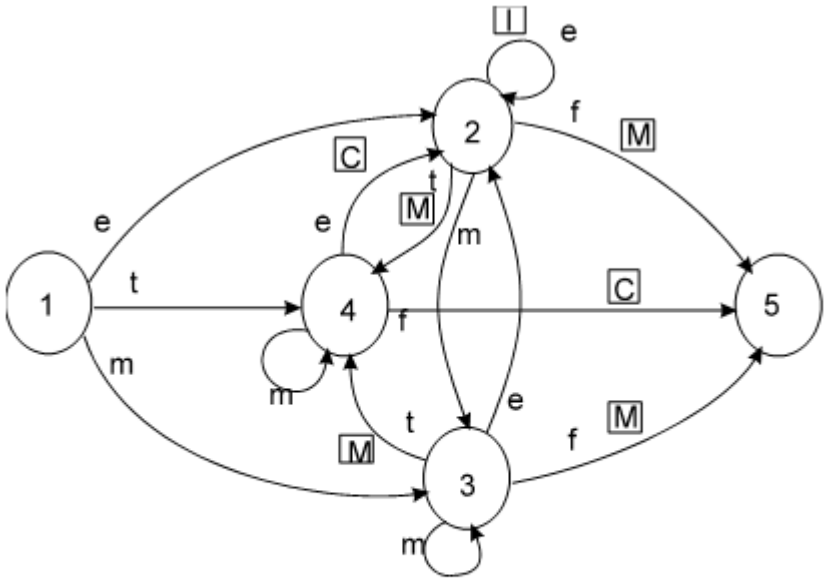
El segundo nivel de clasificación del tiempo de mantenimiento también debe ser asistido por el entorno. El tiempo de pruebas queda claramente definido ya que sería el tiempo de ejecución del programa. El tiempo de modificación correspondería con una edición "activa", es decir, inserción, modificación o supresión de texto. En la Figura 24 se muestra un diagrama de estados que representa el modo en que esta actividad de control de tiempos puede estar integrada en el editor.

Algunos editores que ceden al usuario el control de entrada y salida del modo de edición activa (p.ej. el conocido **vi** de UNIX), no necesitan el sistema antes descrito puesto que el control de tiempos se podría realizar de una forma mucho más sencilla (en todo momento se sabe si el estado es de edición activa o no). Pero éste no es el caso en los editores más actuales y comúnmente utilizados.

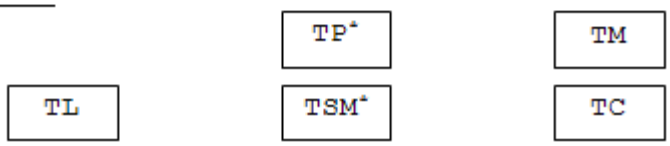
3) Gestión de base de datos.

Para el tratamiento de tan grande cantidad de información histórica es necesario disponer de una herramienta de gestión de bases de datos. El entorno de programación debe incorporar una interfaz con dicha herramienta, que permita:

- Incorporar en la base de datos las medidas obtenidas (tanto del producto como del proceso de producción) a las tablas del proyecto en curso y a las tablas históricas.



Registros:



TP : Contador parcial de tiempo.
TSM: Contador del tiempo sin modificar.
TM : Registro del tiempo de modificación.
TC : Registro del tiempo de comprensión.
TL : Tiempo límite (transcurrido este tiempo sin modificar texto, se entiende que comienza un periodo de tiempo de comprensión).

(*) TP y TSM se autoincrementan con cada segundo transcurrido de edición.

Eventos:
e : Pulsación de tecla de edición (modifica el texto).
m : Pulsación de tecla de movimiento (no modifica).
t : Interrupción por tiempo cuando TSM = TL.
f : Fin de edición.

Acciones:
I : Inicialización del contador TSM (TSM ← 0).
C : Fin de tiempo de comprensión (TC ← TC + TP ; TP ← 0).
M : Fin de tiempo de modificación (TM ← TM + TP ; TP ← 0).

Estados:
1 : Estado inicial (comienzo de edición).
2 : Estado activo (modificación).
3 : Estado transitorio.
4 : Estado pasivo (comprensión).
5 : Estado final (fin de la edición).

Figura 24.- Proceso de control de tiempos para un editor

- Extraer de la base de datos la información que se precise en cada momento, en forma de documentos o informes, o como información de entrada para otras herramientas.

4) Herramienta de cálculo.

Las funciones de ajuste de parámetros del modelo MEDEM y la técnica TAEM, así como la aplicación de los mismos, requieren de una herramienta de cálculo que partiendo de información seleccionada de la base de datos, obtenga dichos resultados (los métodos de cálculo vienen dados en los apartados que se citan):

- Parámetros de la funciones de comprensibilidad, modificabilidad y testeabilidad (apdos. 4.3 y 4.5.2.1).

- Parámetros de la técnica TAEM. (apdos. 4.4 y 4.5.2.1)

- Aplicación del modelo MEDEM (apdos 4.3 y 4.5.2.2)

- Aplicación de la técnica TAEM (apdos 4.4 y 4.5.2.2)

6.1.3 INTEGRACIÓN EN UN ENTORNO DE PROGRAMACIÓN

En este apartado se aborda cuándo y de qué modo han de ser incorporadas las herramientas expuestas en el apartado anterior, a entornos de programación (sin considerar entornos concretos).

El modelo de productividad propuesto afectaría a los entornos de programación clásicos (vease apdo. 2.2.2) en el sentido de incluir nuevas herramientas para extraer las métricas (Véanse Figura 25 y Figura 26). La herramienta de extracción de métricas del producto podría ser independiente, tomando como entrada el fichero del código fuente y grabando la salida en un fichero de métricas.

La herramienta de obtención de métricas del proceso debería estar integrada en el editor, que debería tener las características expresadas en el punto (2) del apartado anterior.

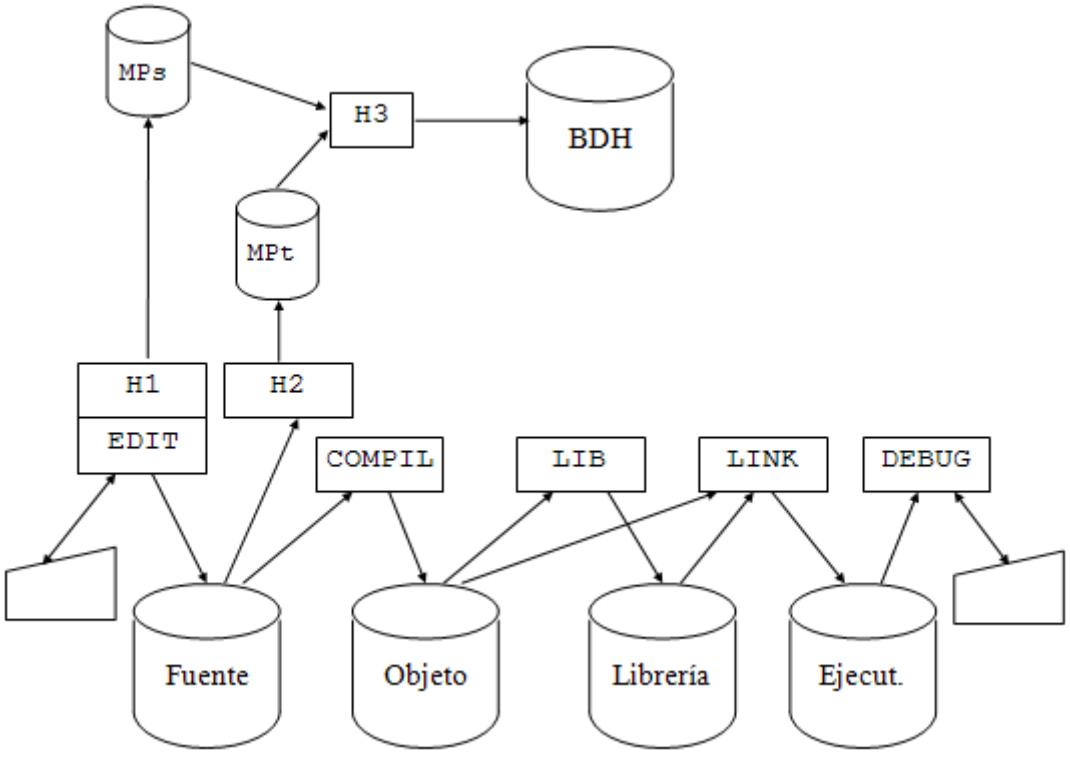
Las demás herramientas se apoyan en la base de datos históricos y en el documento de estimación de los TCAs (tráfico de cambio anual) (vease Figura 26), elaborado tras el diseño preliminar del proyecto.

Denotamos las herramientas nuevas con los siguientes nombres:

H1: Extracción de métricas del proceso.
H2: Extracción de métricas del producto.
H3: Grabación en la B.D.H.
H4: Ajuste de MEDEM y TAEM
H5: Aplicación de la técnica TAEM
H6: Aplicación del modelo MEDEM

Al **comienzo del proyecto** se aplica H4 con objeto de obtener los parámetros de ajuste del modelo de estimación y la técnica de asignación, partiendo de los datos históricos disponibles en ese momento.

Después del **diseño preliminar**, una vez que se dispone de una estimación de los bloques funcionales que van a componer el sistema, así como de su tráfico de cambio anual, se procede a aplicar la técnica de asignación equilibrada de mantenibilidad (TAEM) o herramienta H5.

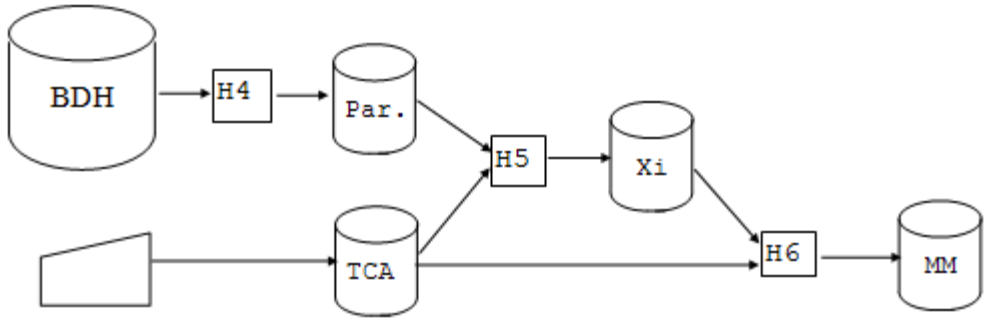


H1 .- Extracción de métricas del proceso.
MPs.- Métricas del proceso.
H2 .- Extracción de métricas del producto.
MPt.- Métricas del producto.
H3 .- Grabación en la B.D.H.

Figura 25.- Integración del modelo de productividad en los entornos clásicos

Inmediatamente después se ha de aplicar el modelo de estimación MEDEM, o herramienta H6, obteniendo así una estimación del esfuerzo en la etapa de mantenimiento.

Durante la **codificación** se ha de tener en cuenta el fichero Xi que contiene los valores de mantenibilidad recomendados. Se realizarán las mediciones pertinentes mediante las herramientas H1 y H2, almacenando la información en la BDH (tabla de programas) a través de H3.



H4.- Ajuste de MEDEM y TAEM.
Par.- Parámetros de MEDEM y TAEM
TCA.- Documento de TCA estimado por programa.
H5.- Aplicación de la técnica TAEM.
Xi.- Valores de mantenibilidad asignados:
X_C , X_M y X_T.
H6.- Aplicación del modelo MEDEM
MM.- Esfuerzo de mantenimiento estimado.

Figura 26.- Integración en el entorno de las herramientas de ajuste y aplicación del modelo MEDEM y la técnica TAEM

En la etapa de **mantenimiento** se aplicarán también las herramientas H1, H2 y H3, esta última sobre la tabla anual de mantenimiento.

En los entornos integrados (vease apdo. 2.2.2 y Figura 27) se podría automatizar el uso de las distintas herramientas del siguiente modo:

(H4) Su ejecución puede ser lanzada, de forma automática, al abrir un nuevo proyecto.

(H5) y (H6) Se ejecutan automáticamente, una vez se han especificado los bloques funcionales que consituyen el

proyecto y sus TCAs, es decir, al dar de entrada esta información en el entorno.

(H1) La herramienta de extracción de medidas del proceso debe estar integrada en el editor, según se ha expresado en el apartado 6.1.2.

(H2) La herramienta "make"[15] podrá incluir en su lista de actividades una llamada para extraer las métricas de mantenibilidad de todos aquellos programas que hayan sido modificados.

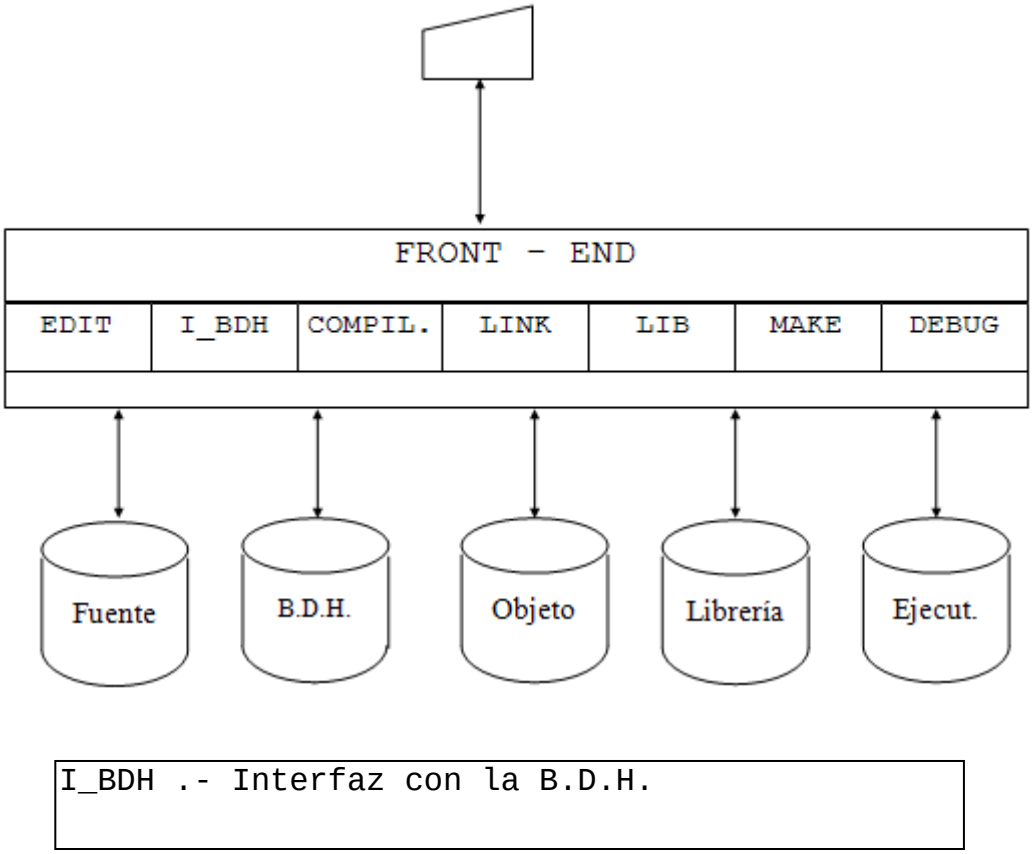


Figura 27.- Inclusión del modelo de productividad en un entorno integrado

(H3) La grabación de las métricas en la BDH puede ser automática e inmediatamente después de su obtención.

[15]"Make" es una herramienta que actualiza los ficheros objetos y ejecutables implicados al modificador código fuente (de acuerdo a un grafo de relaciones que dicha utilidad gestiona en el fichero "makefile").

6.2 TÉCNICA DE MEDICIÓN DE LA COMPRENSIBILIDAD

Un campo de estudio sobre el que no se ha abundado en la tesis, por lo que se considera de interés para trabajos futuros, es el estudio de técnicas que permitan medir directamente la mantenibilidad, en sus tres componentes: comprensibilidad, modificabilidad y "testeabilidad". De este modo, se podrá evaluar la bondad de las estimaciones, sin tener que llegar a la etapa de mantenimiento.

Según se expone en el apartado 2.1.5 la comprensibilidad constituye el principal factor determinante de la mantenibilidad. Seguidamente se propone una aproximación a una técnica para la medición de la comprensibilidad basada en juicio de expertos.

En la medida en que un programa sea más comprensible, será más mantenible. El determinar si un programa es comprensible o no, de forma empírica, es algo bastante subjetivo. El estudio de un programa por parte de una persona, con el fin de comprender su funcionamiento, está sujeto a muchas variables (tiempo empleado en el estudio, grado de conocimientos, estado de ánimo, ...). Para evitar tal subjetividad es preciso utilizar la herramienta **GDC** (Grupo de Decisión Colectiva, ya empleada en el desarrollo de la tesis).

Al finalizar cada una de las etapas de diseño y codificación, se realiza un estudio de desviaciones de los estándares, que ha de ser sometido al juicio del **GDC.** Al mismo tiempo, y considerando el nivel de mantenibilidad requerido para cada módulo, se seleccionarán aquellos módulos cuyo grado de comprensibilidad deba ser analizado por el GDC. Así, la documentación de tales módulos será facilitada a dicho grupo para estudiar su comprensibilidad.

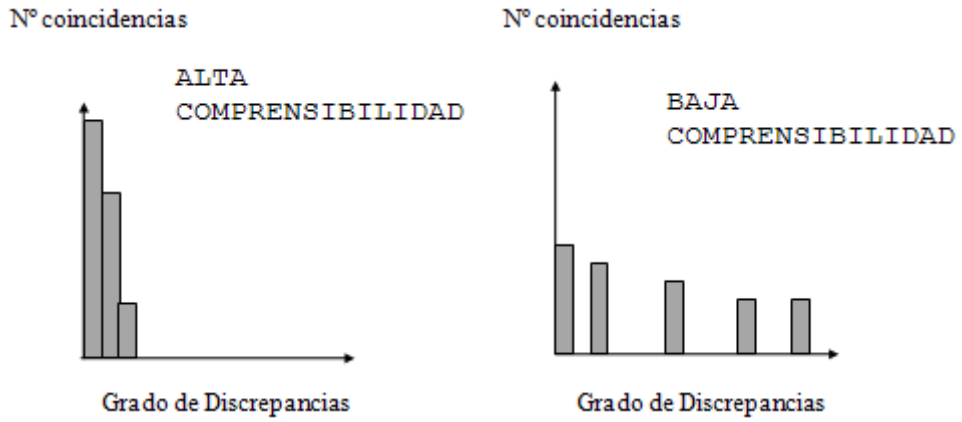


Figura 28.- Analisis de resultados de opiniones de expertos sobre la comprensibilidad

Los resultados del estudio serán analizados, valorando las discrepancias entre opiniones distintas (véase Figura 28)

El procedimiento a seguir para realizar tal estudio, consta de las siguientes fases:

FASE 1.- Selección de expertos: Por cada módulo se realizará una selección de expertos del EDPI a los que se encargará el estudio de comprensibilidad del módulo. Para tal selección se deberán considerar las áreas de conocimiento de cada experto.

Tabla 11.- Tabla de selección de expertos

	Módulo 1	**Módulo 2**	**...**	**Módulo n**
Experto 1				
Experto 2				
...				
Experto m				

A cada experto se le enviará la información necesaria para el estudio. En dicha información se indicará el tiempo aproximado que deben emplear en dicho estudio de comprensibilidad.

FASE 2.- Recogida de informes de expertos: Transcurrido un tiempo, se recogerán los informes de los expertos para su análisis por parte del EDPRAI.

FASE 3.- Agrupación por coincidencias: Por cada módulo estudiado, se agruparán los informes coincidentes, esto es, aquellos que claramente expresen la misma idea.

FASE 4.- Valoración de las discrepancias: Será preciso valorar las discrepancias entre opiniones distintas, según cierta escala.

Tabla 12.- Tabla de valoración de discrepancias

Opiniones	Nº Coincidencias	Valor discrepancías
Opinión 1 (*)	x_1	$v_1=0$
Opinión 2	x_2	v_2
...		
Opinión n	x_n	v_n

(*) Opinión con más coincidencias

FASE 5.- Análisis de resultados: Se deberá medir la dispersión de los resultados. La medida propuesta para ello es la **desviación estándar** de los valores obtenidos en la fase 4.

$$S = \sqrt{\frac{\sum x_i v_i^2}{\sum x_i}}$$

Valores grandes de la desviación estándar (S) indicarán una baja comprensibilidad del módulo.

Tabla 13.- Tabla resultado del estudio de comprensibilidad

Módulo	*Desviación estándar (S)*
Módulo 1	
Módulo 2	
...	
Módulo n	

6.3 TÉCNICA DE SECTORIZACIÓN PARA EL ANÁLISIS DE LOS DATOS HISTÓRICOS

La dificultad de manejar las tablas de la Base de Datos, cuando su tamaño es elevado, constituye un importante problema, sobre todo cuando ha de extraerse información para ser analizada por personas (p.e. en toma colectiva de decisiones han de manejarse informes construidos a partir de los datos históricos).

En este apartado se propone el uso de un algoritmo de sectorización, que permite agrupar “puntos” (registros de las tablas) “próximos” entre sí.

La BDH está compuesta por tablas, que no son más que un conjunto de tuplas de medidas extraidas tanto de la mantenibilidad del producto como de la productividad del proceso.

Una idea interesante como vía de futura investigación sería una técnica de sectorización basada en un algoritmo de J. Litke [LITK84] y empleada en un trabajo anterior del autor [BARR91].

A grandes rasgos, este algoritmo comprende tres pasos:
1) Hacer particiones o sectores de puntos próximos
2) Calcular el centro de gravedad de cada sector

3) Volver al punto (1) con el conjunto de centros de gravedad hasta obtener un conjunto “manejable”.

Para aplicar esta técnica se precisa de una función **distancia** que vendrá dada por la expresión:

$$distancia(j,k) = \sqrt{\left(m_{1j} - m_{1k}\right)^2 + \ldots + \left(m_{mj} - m_{mk}\right)^2}$$

donde m_{ij} (i=1..m) (j=1..N) es la fila j de la métrica m_i .

El criterio “hasta obtener un conjunto manejable” indicado en el punto (3) es bastante subjetivo y se deja libertad para que el GDC lo determine.

Como resultado de aplicar esta técnica a una tabla de la base de datos se obtendría una tabla resumen, con los puntos más representativas, incluyendo una columna con la “densidad del punto”, esto es, el número de puntos reales que representa.

A continuación se da una visión rápida del algoritmo:

En cada iteración, se fija un tamaño del sector (que va aumentando de iteración en iteración), y se construye un conjunto de puntos tratados. Los puntos no tratados se van añadiendo a este conjunto agrupándose con aquellos que estén más próximos, siempre que el tamaño del grupo resultante no supere el tamaño de sector prefijado.

El conjunto de centros de gravedad de cada sector da lugar a un nuevo conjunto de puntos a tratar por la siguiente iteración.

En [BARR91] se presenta con todo detalle este algoritmo.

7 CONCLUSIONES Y TRABAJOS FUTUROS

En este apartado se exponen las conclusiones que se desprenden de la presente tesis. Se comentan las ventajas e inconvenientes del empleo del modelo propuesto. Por otro lado, en el segundo subapartado se presentan las líneas de trabajo futuro, es decir, un conjunto de temas estrechamente relacionados con el área de estudio que podrán ser objeto de futuras investigaciones.

7.1 CONCLUSIONES

El tema central de esta tesis es el estudio de la conexión que existe entre la mantenibilidad de un producto software y la productividad alcanzada durante su mantenimiento. Dos son las principales aportaciones que han sido fruto de este estudio:

1) Modelo de estimación del esfuerzo de mantenimiento.

Se propone un modelo de estimación del esfuerzo que deberá realizarse en la etapa de mantenimiento, obtenido como producto de tres factores: el esfuerzo de desarrollo, el tráfico de cambio anual y un índice de la productividad. Este último factor recibe el nombre de índice de mantenibilidad ya que se construye como una función de tres métricas que miden respectivamente los tres componentes de la mantenibilidad: comprensibilidad, modificabilidad y testeabilidad.

Dado que el valor de las estimaciones depende de la prontitud con que se disponga de ellas, se tiene que el modelo debe aplicarse lo antes posible durante el ciclo de vida del proyecto. Para su aplicación se necesita disponer de una relación de los bloques funcionales que componen el sistema a desarrollar, juntamente con una estimación del tamaño y del tráfico de cambio anual de cada bloque. Esta información puede estar disponible tras la etapa de diseño preliminar, por lo que el modelo se aplica tras dicha etapa. La gran ventaja está en que se puede disponer en los inicios del proyecto de la información que ofrece el modelo, a pesar del inconveniente de tener que realizar estimaciones que dependen de etapas posteriores.

2) Técnica de asignación equilibrada de la mantenibilidad.

Uno de los elementos del modelo de estimación del esfuerzo en mantenimiento es el índice de mantenibilidad, obtenido en función de una métricas de comprensibilidad,

modificabilidad y testeabilidad. Para aplicar el modelo antes de realizar el diseño y la codificación es preciso estimar los valores de dichas métricas. En lugar de estimar, la técnica de asignación equilibrada de mantenibilidad asigna valores óptimos de mantenibilidad a cada bloque funcional, considerando su tráfico de cambio anual así como una estimación del costo de aplicar las características de mantenibilidad obtenida del análisis de datos históricos. Estos valores de mantenibilidad recomendados por la técnica habrán de ser tenidos en cuenta durante las posteriores etapas de desarrollo.

La ventaja principal de esta técnica está en que el esfuerzo de asignación de características de mantenibilidad se optimiza asignando a cada bloque funcional en proporción con su estimación de tráfico de cambio anual, evitando así las inversiones inútiles que no reportan beneficio alguno durante el mantenimiento. El inconveniente está en tener que controlar, durante el desarrollo del producto, los valores de mantenibilidad asignados por la misma. Este inconveniente puede ser paliado con la ayuda de un entorno de programación, que puede ir obteniendo las métricas de mantenibilidad, en tiempo real, durante la edición de los programas.

Otro resultado de la tesis es el control eficiente de las versiones durante la etapa de mantenimiento, lo cual repercute de forma importante en la productividad durante dicha etapa.

Son también fruto de la tesis un estudio de valoración de los factores a considerar en la elaboración de un plan de calidad, y un estudio gráfico de representación conjunta de varios factores de calidad.

Considerado en su conjunto, el modelo de productividad propuesto presenta algunos inconvenientes o restricciones:

- La empresa de software necesita cierta infraestructura para las actividades de decisiones colectivas con juicio de expertos. Son actividades costosas a corto plazo, y en pequeños proyectos pueden resultar poco convenientes.

- Costo inicial de adaptación del entorno y de formación del personal para la aplicación del modelo.

- Los datos históricos son una pieza fundamental en el modelo. Inicialmente, las estimaciones basadas en un conjunto reducido de proyectos pueden resultar poco fiables.

En contrapartida, la gran ventaja de este modelo consiste en que, con su aplicación se dispone de instrumentos de control para estimar el coste y la productividad de la etapa de mantenimiento, que son de enorme valía para la gestión del proyecto. En definitiva se trata de estimar en su justa medida el volumen de ese "iceberg" que R. Canning [CANN72] compara con la etapa de mantenimiento de un proyecto software; y también, en la medida de lo posible, reducir su tamaño mediante una correcta asignación de características de mantenibilidad al producto.

7.2 LÍNEAS DE TRABAJO FUTURO

Los principales aspectos que han quedado sin tratar en la presente tesis y que serán objeto de trabajos futuros están expresados en los siguientes puntos:

1) Ampliación del estudio de métricas de mantenibilidad. Se han propuesto tres métricas para evaluar respectivamente, la comprensibilidad, modificabilidad y testeabilidad. Pero, evidentemente, aún hay mucho que estudiar en este campo, tanto en la programación imperativa tradicional como otros entornos más novedosos: programación declarativa, programación orientada a objetos, etc.

2) Incidencias de la reusabilidad sobre la productividad en la etapa de mantenimiento. Los componentes reusables de software poseen unas características de mantenibilidad y calidad en general que son distintas a los específicamente creados para un proyecto concreto.

3) Estudio de la productividad en la etapa de mantenimiento siguiendo un modelo transformacional de desarrollo de software. El modelo transformacional presenta una problemática distinta en la etapa de mantenimiento, ya que no se realiza sobre el código resultante del proceso de producción sino sobre las especificaciones formales de necesidades.

4) Ampliación del estudio de la integración del modelo de productividad en los entornos de programación. En el apartado 6.1 se expone en qué consiste la integración del modelo en un entorno de programación. Se pretende continuar

en un futuro con este estudio, aplicandolo en algún entorno concreto.

5) Medición directa de las características de mantenibilidad. Se trata de continuar el tema de investigación iniciado en el apartado 6.2 sobre la posibilidad de obtener medidas directas de mantenibilidad (antes de llegar a la etapa de mantenimiento) pudiendo así relacionar dichas medidas con las métricas de mantenibilidad del producto. Esta herramienta se utilizaría como complemento en la actividad de control de niveles de mantenibilidad del modelo propuesto.

6) Tratamiento de grandes bases de datos históricos. El estudio directo de las bases de datos históricos (que alcanzarán tamaños bastante elevados) es impensable sin alguna herramienta que, de forma automática, permita resumir el volumen de información según ciertos criterios. Este estudio ha sido iniciado en el apartado 6.3.

REFERENCIAS BIBLIOGRÁFICAS

[ABDE93] Abdel-Hamid, T. K., "Adapting, correcting and perfecting software estimates: A maintenance metaphor", *Computer*, Marzo 1993, pp. 20-29.

[AGRE86] Agresti, W.W., *New paradigms for software development*, IEEE Computer Society Press, Washington D.C., EE.UU., 1986.

[ALBR79] Albretch,A.J. "Measuring application development productivity", *Proc. IBM Applic. Dev. Symposium*, Monterey, CA (EE.UU.), Octubre 1979, pp. 83-92.

[ALBR83] Albretch,A.J.; Gaffney,J.E. "Software function, source lines of code and development effort prediction: A software science validation", *IEEE Trans. Software Engineering*, Noviembre 1983, pp. 639-648.

[BALZ83] Balzer,R.; Cheatham,T.E.; Green,C., "Software technology in the 1990s: Using a new paradigm", *Computer*, vol. 16, nº 11, Noviembre 1983, pp. 39-45.

[BANK93] Banker,R.D.; Datar,S.M.; Kemerer,C.F.; Zweig, D., "Software complexity and maintenance costs", *Communications of the ACM*, Vol 36, Nº 11, Noviembre 1993, pp. 81-94.

[BARR91] Barranco, M. J., *Desarrollo de una aplicación para la gestión de una empresa dedicada al servicio de pedidos a domicilio*, Trabajo fin de carrera, Facultad de Informática, Universidad Politécnica de Madrid, Julio 1991.

[BARR94] Barranco, M.J.; Granja, J.C.; Martínez, J.A.; González, P. "Control de versiones: Un enfoque práctico". *Novática*, nº 111, Sep-Oct. 1994, pp. 11-17.

[BARR95] Barranco, M.J.; Granja, J.C., "Control of versions: A practical approach oriented to improve the software quality", *Proc. of Software Quality Management (SQM 95)*, Sevilla, Abril 1995, pp. 335-346.

[BELA72] Belady, L.; Lehman, M., "An introduction to growth dynamics", *Statistical Computer Performance Evaluation*, W. Freiberger (Coord.), Academic Press, 1972, pp. 503-511.

[BERN94] Berner, S. "Semantic naming convention and software quality". *4th European Organization for Quality - Software Commitee (EOQ-SC 94)*, Basilea (Suiza), Octubre 1994, pp. 56-65.

[BOEH79] Boehm, B.W. “Software engineering: R&D trends and defense needs”, *Research Directions in Software Technology*, P. Wegner (coord.), MIT Press, 1979, pp. 44-86.

[BOEH81] Boehm, B.W., *Software Engineering Economics*, Prentice-Hall, 1981.

[BOEH88] Boehm, B.W. “A spiral model of software development and enhancement”. *Computer*, vol 21, nº 5, Mayo 1988.

[CANN72] Canning, R., “The maintenance ‘iceberg’”. *EDP Analycer*, vol 10, nº 10, Octubre 1972.

[CHAP88] Chapin, N., “Software maintenance life cycle”, *Proc. Conference on Software Maintenance*, Phoenix, Az. (EE.UU.), 1988. pp. 6-13.

[CLEM89] Clemm, G.M., “Replacing version control with job control”, *Proc. 2nd Intl. Workshop on Software Configuration Management of ACM*, Princeton, Octubre 1989, pp. 162-169.

[CONT86] Conte, S.D.; Dunsmore, H.E.; Shen, V.Y., “Software engineering metrics and models”, *Benjamin/Cummings*, 1986.

[CURT80] Curtis, W. “Management and experimentation in software engineering”. *Proc. of the IEEE*, vol. 68, nº 9, Septiembre 1980.

[EURO94] *Euromethod Concepts Manual 2: Deliverable Model*, Eurometh od Project, 1994.

[FENT94] Fenton, N., “Software measuremente: A necessary scientific basis”, *IEEE Trans. on Software Engineering*, Vol. 20, Nº 3, Marzo 1994, pp. 199-206.

[FROS85] Frost,D. “Software maintenance and modifiability”, *Proc. 1985 Phoenix Conference on Computers and Communications*, Phoenix, Az. (EE.UU.), 1985

[GONZ94] González, P.; Granja, J.C.; Martínez, J.A.; Barranco, M.J. “Estudio de factores de calidad del software partiendo de la ERCU”. *Novática*, nº 111, Sep-Oct. 1994, pp. 18-19.

[GRAN92/1] Granja,J.C. “Contribución al estudio de las técnicas de garantía de calidad”. *Novática*, Vol. XVIII, nº 99. Sep-Oct. 1992.

[GRAN92/2] Granja, J.C., *Aportación a las técnicas de garantía de calidad de software: Su incidencia en la planificación*. Tesis doctoral, Facultad de Informática de Madrid, U.P.M., 1992.

[GRAN95] Granja, J.C.; Barranco, M.J., *Proyectos informáticos*, ISBN: 84-600-8855-3. DL: J-183-94, UNED, 1995.

[HALL94] Hall, T.; Fenton, N., "Implementing software metrics: The critical success factors", *Software Quality Journal*, Vol. 3, 1994, pp. 195-208.

[HARR69] Harr,J. "Programming experience for the no. 1 electronic switching system", 1969.

[HARR90] Harrison, W.; Cook, C., "Insights on improving the maintenance process through software measurement", *Proc. of IEEE Conference on Software Maintenance*, San Diego, CA. (EEUU), Noviembre 1990, pp. 37-46.

[HEND87] Henderson, P.B.; Notkin, D., "Guest editor's instruction: Integrated design and programming environments", *Computer*, vol. 20, nº 11, 1987, pp. 12-16.

[HENR91] Henry, S., Wake. S., "Predicting maintainability with software quality metrics", *Journal of Software Maintenance: Research and Practice*, Vol. 3, 1991, pp. 129-143.

[JONE86] Jones,C., *Software Productivity*, McGraw-Hill, 1986.

[KAFU87] Kafura, D.; Reddy, G., "The use of software complexity metrics in software maintenance", *IEEE Trans. on Software Engineering*, SE-13, Nº 3, 1987, pp. 335-343.

[LIE89] Lie, A.; Conradi, R.; Didriksen, T.M.; Karlsson, E-A. "Change oriented versioning in a software engineering database", *Proc. of 2nd. Intl. Workshop on Software Configuration Management of ACM*, Vol. nº 17, nº 7, Noviembre 1989. pp. 56-65.

[LITK84] Litke, J.D. "An improved solution to the travelling salesman problem". *Communications of the ACM*, Vol. 27, Nº 12, Diciembre 1984, pp. 1227-1236.

[LUDE94] Ludewing, J. "People make quality happen (or don't)" *4th European Conference on Software Quality (EOQ-SC'94)*, Basilea, Suiza, Octubre 1994, pp. 11-21.

[MART83] Martin, J., McClure, C., *Software maintenance: The problem and its solution*, Englewood Cliffs, NJ: Prentice-Hall, 1983.

[MART94] Martínez, J.A.; Granja, J.C.; Barranco, M.J.; González, P., "Estudio gráfico para el análisis de la calidad del software", *Novática*, nº 111, Sep-Oct. 1994, pp. 7-10.

[MCCA76] McCabe, T.J. "A Software complexity measure". *IEEE Trans. Software Engineering*, vol. 2, Diciembre 1976, pp. 308-320.

[MCCA77] McCall, J.A.; Richards, P.K.; Walters,G.F., *Factors in Software Quality*, Vols. I-III, Rome Air Development Centre, 1977.

[MEND93] Mendes-Moreira, H.M.C.L.; Davies, C.G. "SUE: A software understanding environment to support software maintenance activities", *The Journal of Knowledge Engineering*, Vol.6, nº 1, 1993, pp. 1-11.

[ORLA92] Orlandi, E. "Software quality economics". *3rd European Conference on Software Quality*, Madrid, Noviembre 1992.

[PARN79] Parnas, D. "Designing software for ease extension and contraction", *IEEE Trans. on Software Engineering*, Vol. 2, 1979.

[PEER81] Peercy, D. "A software maintainability evaluation methodology", *IEEE Trans. on Software Engineering*, Vol SE-7 Nº 7, 1981, pp 343-351.

[PICK93] Pickard, M.M.; Carter, B.D. "Maintainability: What is it and how do we measure it?", *Software Engineering Notes*, Vol 18, nº 3. Julio 1993, pp. A36-A39.

[PLAI93] Plaice, J.; Wadge, W. "A new approach to version control", *IEEE Trans. on Software Engineering*, vol. 19, nº 3, Marzo 1993, pp. 268-276.

[PRES92] Pressman, R.S. *Ingeniería del Software: Un enfoque práctico*, McGraw-Hill, 3ª ed., 1992.

[ROCH94] Roch, J.M. "Software metrics and measurement principles", *Software Engineering Notes*, vol 19, nº 1, Enero 1994, pp. 77-85.

[SCHÄ84] Schäfer, H., "Metrics for Maintenance Management", *Proc. of COMPAS'84*, 1984.

[SHEP94] Shepperd, M.; Ince, D. C., "A critique of three metrics", *Journal of Systems and Software*, Nº 26, 1994, pp. 197-210.

[SPEM88] ESPRIT, *SPEM Project Documentation*, 1988.

[SPEM90A] ESPRIT-SPEM, *Development Documentation*, 1990.

[SPEM90B] ESPRIT-SPEM, *SPEM Model Building SPEM Prototype Specification*, 1990.

[STAR94] Stark, G. E.; Kern, L. C.; Vowell, C. W., "A software metric set for program maintenance management",

Journal of Systems and Software, Nº 24, 1994, pp. 239-249.

[SWAN76] Swanson, E.B., "The Dimensions of Maintenance", *Proc. 2nd Intl. Conf. Software Engineering*, IEEE, Octubre 1976, pp. 492-497.

[SWAR82] Swartout, W. y Balzer, R., "On the inevitable intertwining of specification and implementation", *Communications of the ACM*, Vol. 25, Nº 7, Julio 1982, pp. 438-440.

[VELA90] Velázquez, A., "Motivación de la programación transformacional", *Factultad de Informática de la Universidad Politécnica de Madrid*, 1990.

[WALL94] Wallmüller E., Software Quality Assurance: A Practical Approach", *Prentice Hall*, 1994.

[WALS77] Walston, C.; Felix, C., "A method of programming measurement and estimation", *IBM Systems Journal*, Vol. 16, Nº 1, 1977.

[WANG84] Wang, A.S., *The estimation of Software Size and Effort: An Approach based on the Evolution of Software Metrics*, 1984.

[WIEN84] Winer, R.; Sincovec, R., *Software Engineering with Modula-2 and Ada*, Wiley & Sons, 1984.

PUBLICACIONES REALIZADAS SOBRE LA TESIS

- "Control de versiones: Un enfoque práctico". Novática, nº 111, Sep-Oct., 1994. (Otros autores: J.C. Granja, J.A. Martínez y P. González).

- "Estudio gráfico para el análisis de la calidad del software". Novática, nº 111, Sep-Oct., 1994. (Otros autores: J.C. Granja, J.A. Martínez y P. González).

- "Estudio de factores de calidad del software partiendo de la ERCU". Novática, nº 111, Sep-Oct., 1994. (Otros autores: J.C. Granja, J.A. Martínez y P. González).

- "Proyectos Informáticos". ISBN: 84-600-8855-3. DL: J-183-94. UNED, 1995. (Otro autor: J.C. Granja).

- "Control of versions: a practical approach oriented to improve the software quality", Proc. of Software Quality Management (SQM 95), Sevilla, Abril 1995. (Coautor: J.C. Granja).

- "Modernización de las administraciones públicas: aprovechamiento de los recursos informáticos, una aportación a las técnicas de evaluación de la productividad de software, el enfoque del control de versiones", TECNIMAP'95, Palma de Mayorca, Mayo 1995. (Coautor: J.C. Granja).

- "Arquitectura cliente/servidor: Incidencia del control de versiones en el análisis de costos", CIL'95, Barcelona, Junio 1995. (Coautor: J.C. Granja).

- "A contribution to the study of improvement of the productivity in software products", SIGCSE Bulletin, Septiembre, 1995. (Coautor: J.C. Granja).

- "Environment Information Systems", Database and Expert Systems Applications (DEXA 95), Londres, Septiembre 1995. (Coautor: J.C. Granja).