Facultad 3

Título: Análisis de Metodologías para la estimación de tamaños de productos software.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Anner Michel Agüero Borrego

Tutor: Ing. Carlos Hidalgo García

Ciudad Habana, Junio 2007
"Año 49 de la Revolución"

DECLARACIÓN DE AUTORÍA

Yo: **Anner Michel Agüero Borrego** declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 y al Departamento de Informatización de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ___ días del mes de _____ del año _____.

_____	_____
Anner Michel Agüero Borrego	Tutor: Ing. Carlos Hidalgo García

OPINIÓN DEL TUTOR DEL TRABAJO DE DIPLOMA

Sobre el Trabajo de Diploma Presentado para Optar por el Título de Ingeniero en Ciencias Informáticas

Título: Análisis de Metodologías para la estimación de tamaños de productos software.
Autor: Anner Michel Agüero Borrego.

El tutor del presente trabajo de diploma considera que durante su ejecución el diplomante ha mostrado total independencia, creatividad y responsabilidad en la solución a situaciones inéditas para él. Ha aplicado conocimientos adquiridos en su formación, demostrando dominio e integralidad en el manejo de técnicas de investigación y análisis de problemáticas.

El documento presenta calidad, es de utilidad y cumple con los requisitos necesarios y objetivos trazados.

La aplicación de las experiencias expresadas en el documento servirán de gran ayuda para guiar la producción de software en la facultad.

Por todo lo anteriormente expresado se puede plantear que el diplomante ha cumplido los objetivos propuestos, por lo que se le propone al tribunal otorgarle el título de Ingeniero en Ciencias Informáticas, con una calificación máxima de _____ puntos además de que este trabajo se pueda presentar en eventos científicos y publicarlo.

_____ _____

Ing. Carlos Hidalgo García Fecha

"Tan sólo hace falta una pequeña idea, para hacer un gran sueño realidad."

Fidel Castro Ruz.

AGRADECIMIENTOS

Mis agradecimientos a todas las personas que de alguna forma contribuyeron a la realización de este trabajo.

Colaboración de la:
Instructora Daira Pérez Serrano
Universidad de Ciencias Informáticas (UCI)

Colaboración del:
Profesor Yoan Martínez Márquez
Universidad de Ciencias Informáticas (UCI)

Ayuda de:
Juan Pablo Giraldo Rendón
Ingeniero de Sistemas
UNIVERSIDAD DE MANIZALES

Opinión y disposición de ayuda de:
Dr. Oscar San Juan
Universidad Pontificia de Salamanca

A mi Tutor, por su paciencia y atención:
Ing. Carlos Yasmany Hidalgo García
Universidad de Ciencias Informáticas (UCI)

Al estudiante Yoandy Lichilín Ríos por su colaboración:
Universidad de Ciencias Informáticas (UCI)

Al estudiante Renier Jorge Telles
Universidad de Ciencias Informáticas (UCI)
Por su ayuda y colaboración.

Al Profesor Pedro Yobanis Piñero Pérez
VICEDECANO PRODUCCION INVESTIGACION
Universidad de Ciencias Informáticas (UCI)
Por su ayuda, atención e indicaciones.

Muchas gracias a todos.

DEDICATORIA

A mi madre:

Miriam Paulina Borrego Acosta.

RESUMEN

El presente trabajo tiene como objetivo contribuir al establecimiento de condiciones para la estimación de tamaños de productos de software en la Universidad de Ciencias Informáticas (UCI). Para esto se presenta un estudio de los aspectos generales de las estimaciones del software y se describen los métodos de estimación del tamaño del software más importante, además se hace una propuesta para modelar la estructura del software a estimar, se definen un conjunto de clases, y finalmente se muestran varios modelos de clases para la cuenta de puntos de función.

TABLA DE CONTENIDOS

INTRODUCCION

"La estimación del tamaño del software a desarrollar representa el primer reto importante del planificador de proyectos"

R.S. Pressman

La estimación es más un arte que una ciencia, no existen métodos de estimación completamente exactos. Lo único que puede mejorar la forma de hacer estimaciones es la disciplina, el conocimiento y las habilidades de la persona que las realiza. Es una actividad bastante empírica en la que se aplican técnicas matemáticas, experiencia, e intuición.

Las estimaciones del software se comienzan a realizar en las primeras fases del desarrollo del producto, y se perfeccionan, a partir de su realización reiterada a lo largo de todo el ciclo de vida del proyecto. Las estimaciones más tempranas son menos fiables, pero a medida que vamos obteniendo más información, y se van precisando los detalles del producto a desarrollar, las estimaciones que se realicen serán más exactas.

La estimación del tamaño del software es uno de los aspectos más fundamentales de la planificación de un proyecto. Esta guarda una estrecha relación con el esfuerzo de desarrollo, tiempo, y recursos necesarios para concebir efectivamente el producto de software final. De aquí que sea este un tema de vital importancia, para cualquier organización desarrolladora de software.

Planteamiento del problema

En los proyectos productivos de la Universidad de Ciencias Informáticas (UCI) no se estima el tamaño de los productos de software a desarrollar. Como consecuencia las estimaciones de costes, esfuerzos, tiempos de desarrollo y recursos para los proyectos no son efectivas o no se realizan. Se hace necesario crear un conjunto de condiciones para la realización de estas estimaciones.

La situación anterior origina el siguiente **problema científico:** ¿Cómo contribuir al establecimiento de recursos y condiciones para la realización de estimaciones de tamaños de productos de software en los proyectos productivos de la UCI?

El **objeto de estudio** de este trabajo comprende la estimación de tamaños de productos de software, en el contexto de la gestión, planificación y estimación de proyectos de desarrollo de software.

Se tiene como **objetivo general** contribuir al establecimiento de recursos y condiciones para la estimación del tamaño de los productos de software en la UCI. Y como **objetivos específicos**, proponer estrategias para la estimación de tamaños de productos de software, junto a la realización de un análisis de las principales metodologías para la estimación del tamaño de productos software, con el fin de establecer una base teórica para la futura implementación de métodos y técnicas con este propósito.

El **campo de acción** comprende los métodos, técnicas y procedimientos para la estimación de tamaños de productos de software, desde una perspectiva para su potencial automatización.

La hipótesis planteada consiste en que si se proponen estrategias para la estimación del tamaño de productos de software, estableciéndose todo el

basamento teórico para la implementación de algoritmos, se establecerá un punto de partida en la creación de un conjunto de condiciones necesarias para llevar a cabo estas actividades en la UCI.

Se identifican las **variables independientes**: estudio de estrategias para la estimación de tamaños de productos de software, con vista a la futura programación de algoritmos de estimación. La **variable independiente** corresponde a la creación de condiciones necesarias para la realización de estimaciones del tamaño del software en la UCI.

Dentro de las **tareas a desarrollar** se cuentan el estudio de técnicas y métodos para la estimación de tamaños de productos de software, y la búsqueda de opinión de profesionales, para la posterior programación de clases y algoritmos para estas estimaciones.

El trabajo aquí expuesto se ha llevado a cabo a partir de la aplicación de varios **métodos científicos de investigación.**

Están presentes varios métodos teóricos. El **método analítico-sintético** se aplicó a través de la búsqueda de información bibliográfica en diferentes fuentes, documentos en Internet, etc. El **método inductivo-deductivo** se aplicó en el análisis de la información obtenida, a partir de la adaptación y generalización de los contenidos. La **modelación** se refleja en el tratamiento dado a los métodos de estimación.

Se utiliza la **entrevista y consulta** a profesionales de la Informática como único método empírico de la investigación.

CAPITULO 1: ASPECTOS GENERALES SOBRE LA ESTIMACION

1.1. La estimación de tamaños de productos de software

Al comienzo de un proyecto de software los requerimientos del producto pueden ser entendidos pero en total poco se sabe del producto en sí mismo. El problema de la estimación consiste en estimar el tamaño más probable para el producto final(*Estimating of Software Size - Part I* 2003).

La exactitud de las estimaciones depende, entre otras cosas, de la fase del ciclo de vida del proyecto donde se realicen. La Figura 1.1 muestra como las estimaciones realizadas al inicio de un proyecto de software pueden oscilar en un rango de cuatro veces por encima y por debajo del valor real. Esto se debe a la gran incertidumbre que se tiene en ese momento sobre la naturaleza del producto(MORENO).

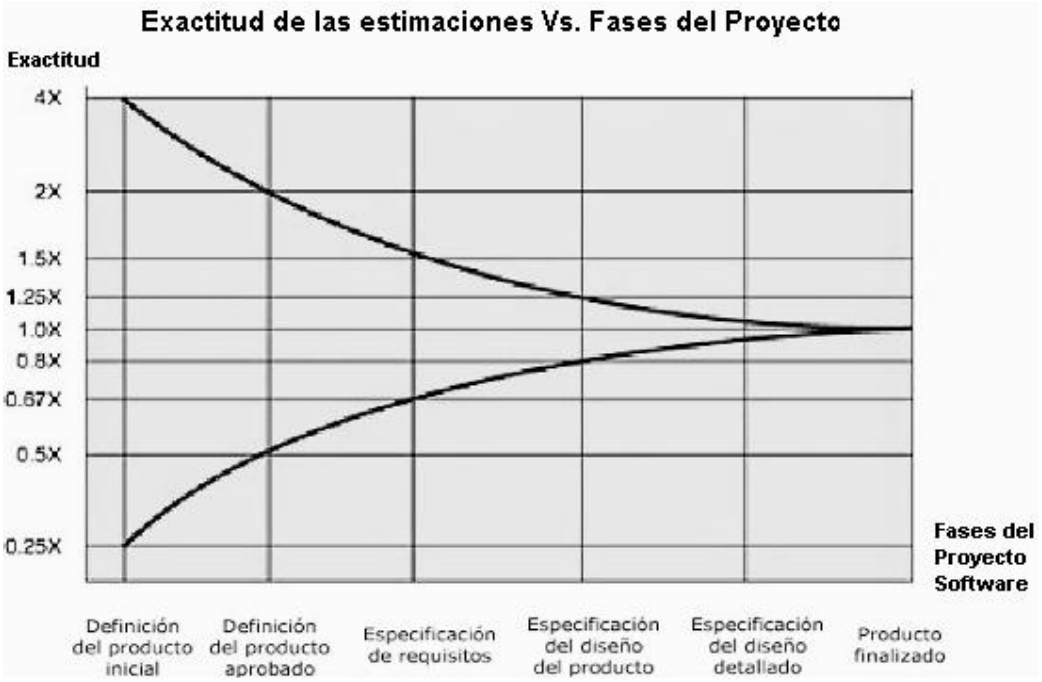

Figura 1.1.Exactitud de las estimaciones a lo largo del desarrollo del proyecto. Adaptado de(MCCONNELL 1996).

En la etapa inicial del proyecto la única vía para realizar estimaciones es por analogía con proyectos anteriores.

De manera general, todos los métodos de estimación utilizan datos históricos de proyectos pasados para estimar el tamaño del nuevo software a desarrollar, ver Figura 1.2(*Estimating of Software Size - Part I* 2003).

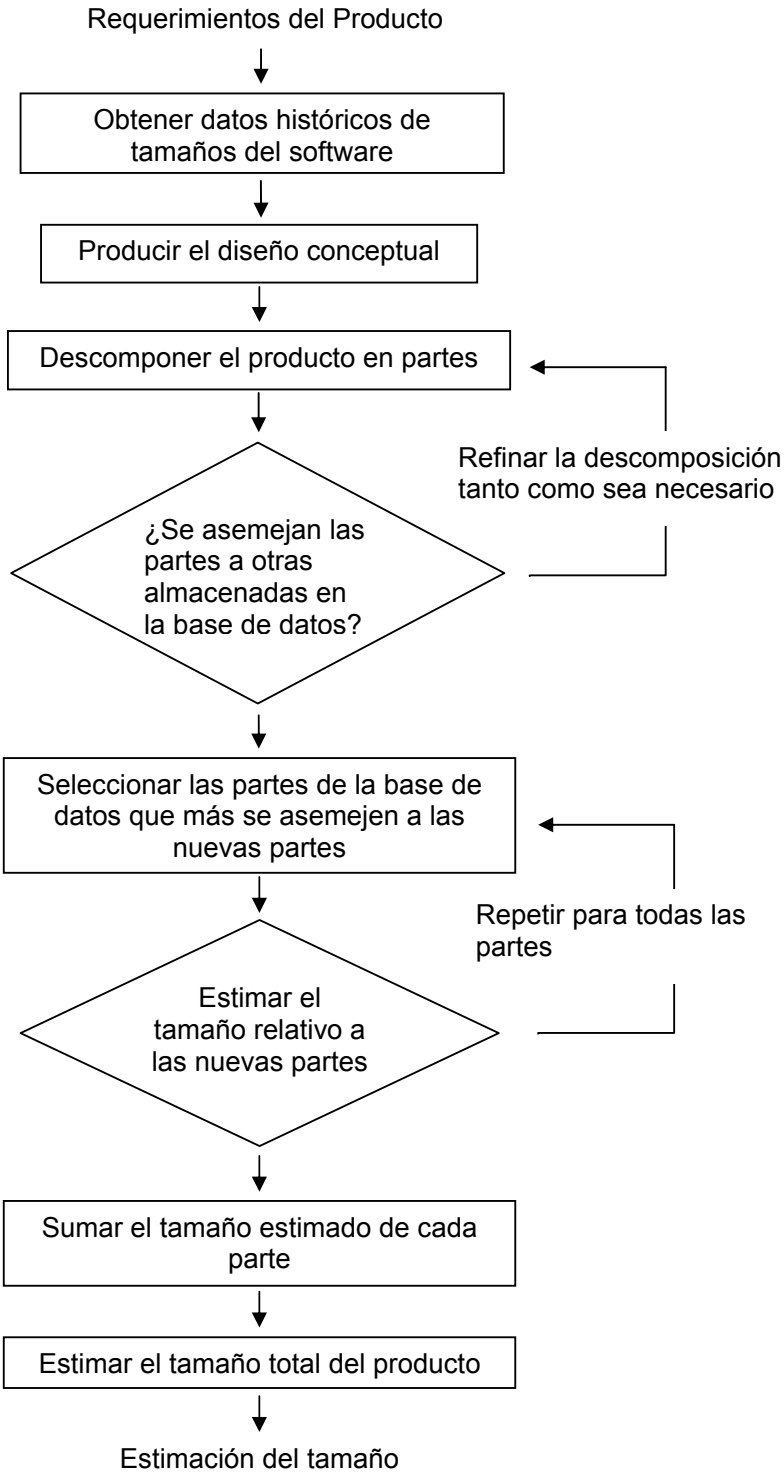

Figura 1.2. Vista general de los procedimientos de estimación del tamaño del software. Descomposición del software y comparación con datos históricos.

1.1.1. Atributos comunes de las estimaciones

Se han desarrollado varias técnicas de estimación para el desarrollo de software, aunque cada una tiene sus puntos fuertes y sus puntos débiles, todas tienen en común los siguientes atributos(*Fundamentos de Ingenieria de Software. Módulo 2* 2007):

- Se han de establecer de antemano el **ámbito del proyecto**[1].
- Como bases para la realización de estimaciones se usan **métricas del software** de proyectos pasados.
- El **proyecto se desglosa en partes** más pequeñas que se estiman individualmente.

1.1.2. Técnicas comúnmente utilizadas

Para la estimación, existen cuatro técnicas básicas y comunes(MORENO):

1. La **opinión de los expertos**; Esta técnica se basa en la experiencia profesional de los participantes en el proyecto de estimación.

2. La **analogía**; Es una aproximación más formal que la experiencia de los expertos y se basa en la comparación directa de uno o más proyectos pasados. La estimación inicial se ajusta dependiendo de las diferencias entre el proyecto pasado y el nuevo.

3. La **descomposición**; Consiste en la descomposición de un producto en componentes más pequeños, o descomponer un proyecto en tareas de nivel inferior. La estimación se hace a partir del esfuerzo requerido para producir los

[1]En este contexto el término *proyecto* puede hacerse corresponder con *producto*.

componentes más pequeños o para realizar las tareas de nivel inferior. La estimación global del proyecto resultará de sumar las estimaciones de los componentes.

4. Las **ecuaciones de estimación**[2]; Son fórmulas matemáticas que establecen la relación de algunas medidas de entrada (que normalmente es la medida del tamaño del producto) y determinan el esfuerzo que se requerirá.

1.1.3. Métodos de estimación

Jerzy Nawrocki propone la aplicación de los siguientes métodos de estimación del tamaño del software según la fase del proyecto(NAWROCKI 1999):

Después del análisis de los requerimientos:

- Puntos de Función

Después de un diseño de alto nivel:

- Método Wideband-Delphi
- Método Difuso de Putnam
- Método de Componentes Estándar

Después de un diseño de bajo nivel:

- Método PROBE

1.2. Medidas del tamaño del software

Se pueden definir un conjunto de **atributos** para medir el **tamaño del software(*Medición del Software*)**:

- **Longitud:** tamaño físico del producto.
- **Funcionalidad:** funciones que proporciona el producto al usuario.
- **Complejidad** (de tiempo o espacio): recursos necesarios (de tiempo o memoria de ordenador) para implementar una solución particular.

[2] Esta técnica se relaciona directamente con las estimaciones del tamaño, pero no se utiliza para estimar el tamaño del software.

1.2.1. Líneas de Código

El número de líneas de código (LOC) es la medida más usada para medir la longitud del código fuente.

Se han realizado muchas propuestas para contarlas. La más extendida es la que no contabiliza las líneas comentadas ni en blanco. La abreviatura que se usa para estas líneas es NCLOC o ELOC (effective lines of code).

Es útil medir por separado las líneas comentadas (CLOC) para calcular esfuerzo, productividad, etc. La longitud total será:

$$LOC = NCLOC + CLOC$$

También puede se útil calcular la densidad de comentarios:

$$CLOC/LOC$$

Para propósitos tales como la prueba es importante conocer cuanto código ejecutable se produce, para ello se mide el número de sentencias ejecutables (ES), ignorando los comentarios, declaraciones de datos y cabeceras.

Otra propuesta consiste en contabilizar únicamente el código entregado al cliente. Se cuenta el número de DSI (delivered source instruction) que incluye las declaraciones de datos, las cabeceras y las instrucciones fuente.

Algunas ventajas del uso de las LOC son(E-CLASES):

- Son fáciles de calcular.
- Muchos modelos de estimación de software usan LOC o KLOC como datos de entrada.
- Existen un amplio conjunto de datos y literatura basados en LOC.

Inconvenientes(E-CLASES):

- Son dependientes del lenguaje de programación.
- Perjudica a los programas cortos pero bien diseñados.
- Su uso en estimación es difícil porque hay que estimar las LOC a producirse mucho antes de que se complete el análisis y el diseño.

Adaptado de (*Medición del Software*).

1.2.2. Puntos de Función

Atendiendo a los inconvenientes presentados por las Líneas de Código (LOC) se ha pretendido establecer una medida del tamaño independiente de la tecnología. Este es el caso de la métrica de Puntos de Función, una medida de la funcionalidad aportada por un sistema o aplicación informática que se determina a partir de ciertas características del dominio de información del software[3].

Un aspecto importante de esta métrica es que adopta una perspectiva desde el punto de vista del usuario sin tener en cuenta los detalles de como se implementará el sistema[4].

La métrica de PF se ha establecido como un estándar de la industria del software, permitiendo la comparación de múltiples aspectos entre diferentes proyectos desarrolladores.

Existen varias metodologías para el conteo de los PF la más popular de las cuales es la mantenida por el Internacional Function Points Users Group (IFPUG)(WIKIPEDIA 2007c).

[3] Para una exposición más detallada véase 2.8 del Capítulo 2.
[4] El término *sistema* se utiliza como sinónimo de *software*.

Esta métrica es muy criticada. Las críticas hablan de dificultades para su aplicación, la falacia de sus fundamentos, su no relación con aspectos concretos del software, etc.

Existe una gran controversia entre que medida utilizar, las orientadas al tamaño, LOC, o las orientadas a la funcionalidad, PF. A continuación la Tabla 1.2.2 muestra una relación entre las Líneas de Código (LOC) y los Puntos de Función (PF). Esta relación depende del lenguaje de programación y de la calidad del diseño(PRESSMAN 2002).

Lenguaje (o entorno de programación)	**LOC/PF**	**Lenguaje (o entorno de programación)**	**LOC/PF**
4GL	40	FoxPro 2.5	34
Ada 83	71	Generador de Informes	80
Ada 95	49	Hoja de Cálculo	6
APL	32	Java	53
BASIC - compilado	91	Modula 2	80
BASIC - interpretado	128	Oracle	40
BASIC ANSI/Quick/Turbo	64	Oracle 2000	23
C	128	Paradox	36
C++	29	Pascal	91
Clipper	19	Pascal Turbo 5	49
Cobol ANSI 85	91	Power Builder	16
Delphi 1	29	Prolog	64
Ensamblador	320	Visual Basic 3	32
Ensamblador (Macro)	213	Visual C++	34
Forth	64	Visual Cobol	20
Fortran 77	105		

Tabla 1.2.2. Relación entre LOC y PF para varios lenguajes, o entornos, de programación[5](*Cálculo de PF*).

En el siguiente capítulo se exponen algunas de las metodologías más importantes para el cálculo de los puntos de función, véase la sección…

1.2.3. Documentación

Esta métrica da una medida de la magnitud de la documentación asociada al producto de software. Teniendo en cuenta las medidas definidas anteriormente pudieran derivarse métricas compuestas como:

- Páginas de Documentación/KLOC (miles de líneas de código).
- Páginas de Documentación/PF

1.2.4. La Métrica Bang

La Métrica Bang es una métrica a partir de la cual se pude derivar una indicación del tamaño del software a implementar como consecuencia del Modelo de Análisis (MA).

Para calcular la Métrica Bang, el desarrollador de software debe evaluar primero un conjunto de primitivas (elementos del MA que no se subdividen más en el nivel de análisis) Las primitivas se determinan evaluando el MA y desarrollando cuentas para varios elementos(*Métricas en el desarrollo del Software*).

[5] La presente tabla es una adaptación de la tabla mostrada en *(Cálculo de PF)*. Los valores mostrados son promedios recomendados. El equipo de trabajo que implemente esta estrategia deberá ajustar estos valores basado en datos históricos de su desempeño real.

1.2.5. Métricas para el Software Orientado a Objetos

Dado que la clase es la unidad fundamental de los sistemas OO la gran mayoría de las métricas están orientan a clases(PRESSMAN 2002).

1.2.5.1. Métricas orientadas a clases

Desde la perspectiva de este trabajo se pueden mencionar dos conjuntos de métricas importantes(PRESSMAN 2002):

La serie de métricas CK. Comprende seis métricas[6] basadas en clases para sistemas OO. Ha sido propuesto por Chidamber y Kemerer.

Ejemplo:

Métodos ponderados por clase (MPC). Asumen que n métodos de complejidad c_1, c_2,... c_n se definen para la clase C. La métrica de complejidad específica que se eligió (por ejemplo, complejidad ciclomática) debe normalizarse de manera que la complejidad nominal para un método toma un valor de 10.

$$MPC = \sum_{i=1}^{n} c_i$$

Métricas propuestas por Lorenz y Kidd. Se separan en cuatro amplias categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas al tamaño para las clases OO se centran en el recuento de atributos y operaciones para cada clase individual, y los valores promedio para el sistema OO como un todo. Las métricas basadas en la herencia se centran en la forma en que las operaciones se reutilizan en la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y los aspectos

[6] Para mayor información véase (Pressman 2002).

orientados al código; las métricas orientadas a valores externos, examinan el acoplamiento y la reutilización.

Ejemplo:

Tamaño de clase (TC). El tamaño general de una clase puede medirse determinando las siguientes medidas:

- el total de operaciones (operaciones tanto heredadas como privadas de la instancia), que se encapsulan dentro de la clase.
- el número de atributos (atributos tanto heredados como privados de la instancia), encapsulados por la clase.

1.2.5.2. Métricas orientadas a operaciones

Ya que la clase es la unidad dominante en los sistemas OO, se han propuesto menos métricas para operaciones(PRESSMAN 2002):

Tamaño medio de operación (TOmedio). Aunque las líneas de código podrían ser usadas como un indicador para el tamaño de operación, la medida LOC adolece de [ciertos problemas]. Por esta razón, el número de mensajes enviados por la operación proporciona una alternativa para el tamaño de operación.

Complejidad de operación (CO). La complejidad de una operación puede ser calculada usando cualquiera de las métricas de complejidad propuestas para el software convencional[7].

Número de parámetros de media por operación (NPmedia). Tan largo como sea el número de parámetros de operación, más compleja será la colaboración entre objetos. NPmedia debe mantenerse tan baja como sea posible.

[7] Se refiere al software no orientado a objetos.

1.2.6. Otras medidas de tamaño

Se pueden establecer otras medidas convenientes del tamaño de un producto de software, teniendo en cuenta sus características. En esto juega un papel fundamental nuestro interés en la medición, como se descompone el producto, qué necesitamos medir, qué necesitamos cuantificar[8]. También es necesario establecer una medida normalizada del tamaño que facilite la comparación entre diferentes productos de software.

1.2.7. Conclusiones parciales

Esta aquí se ha expuesto brevemente un resumen de varios aspectos generales para la estimación de tamaños de productos de software. En el próximo capítulo se describen los principales métodos de estimación del tamaño del producto.

[8] En *Ingeniería del software: un enfoque práctico* (Pressman 2002) se describe el método de desarrollo de métricas *Objetivo Pregunta Métrica (OPM);* que intenta definir un conjunto de métricas apropiadas para el software.

CAPITULO 2: MÉTODOS DE ESTIMACIÓN DEL TAMAÑO DEL SOFTWARE

2.1. Introducción

En el presente capitulo se expone una colección de los métodos más importantes para la estimación del tamaño del software. El contenido ha sido adaptado de diferentes fuentes, tal y como se refiere al final de cada una de las secciones.

2.2. Método Wideband-Delphi

El método Wideband-Delphi fue desarrollado por Rand Corporation y refinado por Boehm.

Se basa en los criterios de varios expertos. Estos ofrecen estimaciones individuales, y luego se utiliza un proceso Delphi para llegar a un consenso en las estimaciones.

Es un método muy importante porque los expertos con frecuencia no están de acuerdo en sus estimaciones.

El procedimiento consta de los siguientes pasos:

1. Se le entrega a un grupo de expertos las especificaciones del producto y un formulario para la estimación.
2. Estos se reúnen para discutir los objetivos del proyecto, ideas, suposiciones, y asuntos de estimación.
3. Luego cada uno de ellos lista anónimamente las tareas del proyecto y el tamaño estimado.
4. Las estimaciones son entregadas al moderador, este tabula los resultados y los vuelve a entregar a los expertos, véase la Figura 2.1.

5. En el formulario únicamente la estimación personal de cada experto se identifica; todas las demás son anónimas.
6. Los expertos se reúnen para discutir los resultados. Cada uno revisa las tareas que ha definido pero no sus estimaciones de tamaño.
7. El ciclo continúa partiendo del paso 3 hasta que las estimaciones converjan dentro de un rango aceptable.
8. El valor estimado se toma como la media de las estimaciones propuestas.

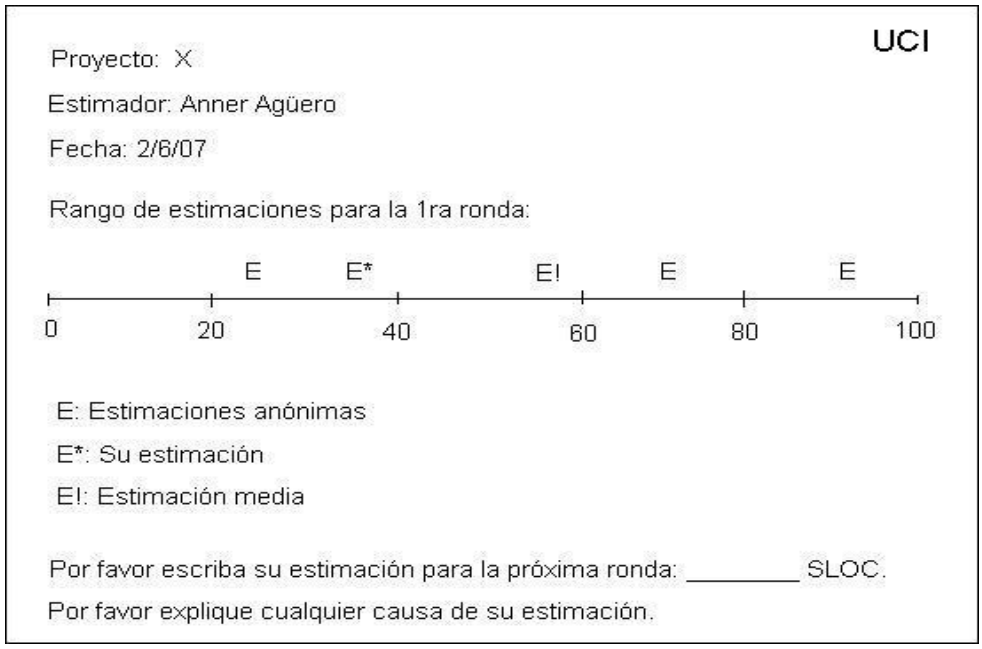

Figura 2.1. Formulario de estimación. Método Wideband-Delphi. Adaptado de (*Estimating of Software Size - Part I* 2003).

Para productos considerablemente grandes, el estimador realiza estimaciones simultáneas para varios componentes del producto. Al final del proceso de estimación, las estimaciones se combinan para producir el total final estimado.

El propósito del método Delphi es compartir puntos de vista. Animando a los participantes a que discutan las tareas del proyecto. Un moderador hábil puede propiciar debates provechosos.

Un aspecto de interés es que la persona que hace una estimación particularmente alta o baja, puede dar una explicación de las razones de su estimación.

Adaptado de(*Estimating of Software Size - Part I* 2003).

2.3. Estimación basada en Componentes Estándar

El método de estimación del tamaño del software basado en componentes estándar, es descrito por Putnam como una vía para hacer estimaciones progresivamente más refinadas utilizando datos históricos.

Se comienza obteniendo datos de tamaños de varios niveles de abstracción del programa, por ejemplo, subsistemas, módulos, y pantallas.

Luego se estima que cantidad de cada uno de estos componentes se necesitará para el nuevo programa. Se estima la mayor y la menor cantidad para cada tipo de los componentes candidatos.

Estas estimaciones se combinan en una función probabilística de tres puntos, como por ejemplo:

Valor estimado = [menor valor + 4*(valor más probable) + mayor valor] /6

En esta ecuación, el valor más probable suele tomarse como el promedio del mayor y el menor valor estimado.

Algunos de los valores utilizados por Putnam para la estimación basada en componentes estándar, son mostrados en la Tabla 2.3.

Componente Estándar	SLOC por Componente	S	M	L	X = (S + 4*M + L) / 6	SLOC
SLOC	1					
Instrucciones objeto	0,28					
Ficheros	2,535	3	6	8	6,17	15,633
Módulos	932	11	18	22	17,5	16,31
Subsistemas	8,175					
Pantallas	818					
Reportes	967	5	9	21	10,3	8,453
Programas interactivos	1,769	2	6	11	6,17	5,963
Programas batch	3,214					
Total						46,359

Tabla 2.3. Valores utilizados por Putnam para la estimación basada en componentes estándar.

Adaptado de(*Estimating of Software Size - Part I* 2003).

2.4. Método difuso de Putnam

Este enfoque utiliza las técnicas aproximadas de razonamiento que son la piedra angular de la lógica difusa. Para aplicar este enfoque, el planificador debe identificar el tipo de aplicación, establecer su magnitud en una escala cuantitativa y refinar la magnitud dentro del rango original. Aunque se puede utilizar la experiencia personal, el planificador también debería tener acceso a una base de datos histórica de proyectos para que las estimaciones se puedan comparar con la experiencia real(PRESSMAN 2002).

2.4.1. Aplicación a la estimación del tamaño del programa

El método difuso de Putnam se fundamenta en que las estimaciones que se requieren no necesitan ser exactas sino convenientes, a un nivel requerido. Utiliza datos históricos de tamaño de proyectos.

Se definen una serie de rangos de tamaños basado en la información histórica disponible, ver Tabla 2.4.1.

Rangos	Bajo	Medio	Alto
Muy pequeño	1 000	2 000	4 000
Pequeño	4 000	8 000	16 000
Mediano	16 000	32 000	64 000
Grande	64 000	128 000	256 000
Muy grande	256 000	512 000	1 024 000

Tabla 2.4.1. Rangos de tamaño básicos. Los valores pueden asumirse como medidas de Líneas de Código (LOC).

Teniendo en cuenta el valor más grande (L), y el más pequeño (S) de tamaño de programa, se procedería a encontrar los valores extremos de cada rango de tamaño: A, B, C, D. Esto se hace de tal manera que L, A, B, C, S, forman una progresión geométrica:

$A/S = B/A = C/B = D/C = L/D = p$

$L/S = p^5$

$p = (L/S)^{0.2}$

Por ejemplo, si S= 1 000 y L= 1 024 000, entonces p=4.

2.4.2. Observaciones

- Se requiere disponer de gran cantidad de datos históricos de un gran número de proyectos.
- No se recomienda cambiar los rangos definidos, sino añadir nuevos rangos cuando sea necesario.

Esta misma técnica puede ser utilizada para estimar otros elementos de tamaño del software. Por ejemplo: la documentación, otras midas de tamaño convenientes, etc. De esta manera se tendría una tabla de rangos para cada aspecto considerado.

Adaptado de(NAWROCKI 1999).

2.5. Método PROBE (Proxy-Based Estimating)

Un proxy es un sustituto, medida o estimador, de la unidad de tamaño, que proveniente de elementos del producto; por ejemplo clases u objetos (y eventualmente métodos)(*1*).

Otros ejemplos de proxys son las pantallas, archivos, scripts, y puntos de función.

El método PROBE se sitúa en el contexto de las fases 0 y 0.1 del Proceso de Software Personal (PSP)(*Capítulo 3. PSP 0 y PSP 0.1*), y utiliza objetos como proxys.

En el método las medidas de tamaño de los objetos son normalizadas. Los objetos se organizan en categorías, para cada una de las cuales se definen rangos de tamaño, tal como propone el método difuso de Putnam en la sección 2.4.

La Figura 2.5 muestra un gráfico de los pasos que componen el método PROBE.

Se recomienda consultar el texto original donde se describe este método(HUMPHREY 1995).

2.5.1. Realizar un Diseño Conceptual

Para que la estimación del tamaño refleje apropiadamente el producto a construir se debe comenzar con un diseño conceptual. Este diseño establece una aproximación preliminar de diseño y nombra los objetos del producto y sus funciones.

No se pretende realizar un diseño completo pero si postular los objetos que se necesitaran y las funciones que realizarán.

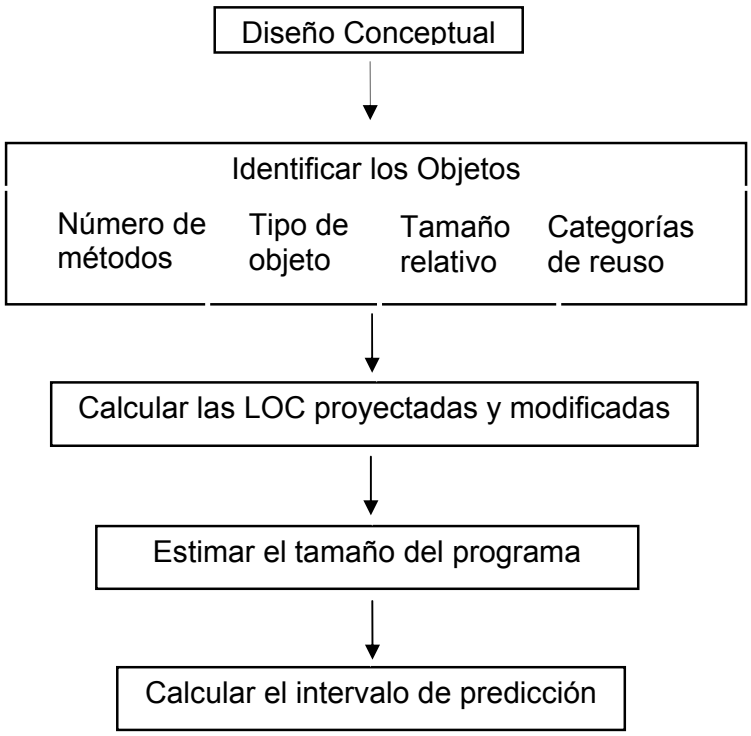

Figura 2.5. Gráfico de los pasos del método PROBE.

Cuando se están estimando productos relativamente pequeños se puede producir un diseño conceptual directamente. Para productos más grandes será necesaria una técnica de diseño que permita subdividir el producto. Si alguna o todas las partes resultantes son aún demasiado grandes para permitir un diseño conceptual suficientemente detallado, entonces será necesario refinar la

partición aun más. Se continua el proceso de refinamiento hasta alcanzar un nivel en el cual se puedan describir las funciones del producto en términos de objetos, estos últimos similares a los objetos almacenados en la base de datos histórica.

2.5.2. Determinar el tipo de objeto y su tamaño

En el diseño conceptual son nombrados los objetos y se les asigna una categoría. El próximo paso consiste en encontrar los objetos de la base de datos histórica que más se asemejen a los objetos del diseño. Para cada objeto se estima como se compara su tamaño con el tamaño de los objetos de su misma categoría en la base de datos.

2.5.3. Formulario para la estimación

En la aplicación del método PROBE se utiliza un formulario como el que se muestra en la Tabla 2.5.3. Los datos de entrada se describen a continuación:

Programa Base. Al mismo tiempo que se estiman los objetos también se determina el tamaño del programa base (código existente) que se esta mejorando y cualquier cambio que se le realice.

Objetos Reutilizados. Para cada objeto que se planee reutilizar se anota su nombre y tamaño en la sección de objetos reutilizados.

Nuevos de Reuso. Son objetos que se van a desarrollar pero que se piensan reutilizar en el futuro, por lo que serán añadidos a la librería de reutilización.

2.5.4. Cálculo de las LOC Objeto estimadas

La cantidad de LOC Objeto estimadas (E) para el nuevo programa se calculan con la formula: E = BA + NO + M

Estudiante:			Fecha:	
Instructor:			# programa:	
Programa Base				LOC
Tamaño Base (B)				
LOC Eliminadas (D)				
LOC Modificadas (M)				
LOC Proyectadas				LOC
Adiciones Base	Tipo	Método	Tamaño	
…	…	…	…	…
Total de Adiciones Base (BA)				
Nuevos Objetos (NO)	Tipo	Método	Tamaño	LOC (Nuevo Reuso*)
…	…	…	…	…
Total de Nuevos Objetos (NO)				
Objetos Reutilizados				LOC
…				…
Total de Reutilizados (R)				
				LOC
LOC Proyectadas		P = BA + NO		
Parámetro de regresión		β_0		
Parámetro de regresión		β_1		
Estimado de LOC nuevas y cambiadas		$N = \beta_0 + \beta_1*(P + M)$		
Total de LOC estimadas		T = N + B – D – M + R		
Estimadas de Nuevos de Reuso		suma de *LOC		
Intervalo de Predicción		Rango		
Intervalo de Predicción Superior		IPS = N + Rango		
Intervalo de Predicción Inferior		IPI = N - Rango		
Porciento del Intervalo de Predicción		%		

Tabla 2.5.3. Formulario de datos para el método PROBE.

Los términos Adiciones Base (BA), Nuevos Objetos (NO), y LOC Modificadas se definen teniendo como referencia el formulario de datos presentado.

BA: LOC que han sido añadidas al programa de partida.
NO: LOC para nuevos objetos que se desarrollan.
M: LOC que se añaden al programa de partida (código existente) como parte de una modificación.

2.5.5. Regresión lineal

¿Cómo se relaciona históricamente el tamaño del programa con las LOC Objeto estimadas?

Ejemplo: el tamaño del programa es históricamente un 25% mayor que el total de LOC estimadas de los objetos que contiene.

El estimado del programa se determina aplicando regresión lineal sobre los datos históricos.

Para los parámetros de regresión, β_0 y β_1, se utiliza la siguiente ecuación:

$$\beta_1 = \frac{\sum_{i=1}^{n} x_i * y_i - n * x_{prom} * y_{prom}}{\sum_{i=1}^{n} x_i^2 - n(x_{prom})^2}$$

$$\beta_0 = y_{prom} - \beta_1 * x_{prom}$$

La fórmula de regresión lineal es:

Tamaño del programa = β_0 + E* β_1

, donde E son las LOC Objeto estimadas.

2.5.6. Intervalo de predicción

Una vez hecha una estimación se necesita medir su calidad.

El intervalo de predicción se calcula utilizando datos históricos y la distribución probabilística t de Student.

Este intervalo da el rango alrededor de la estimación en el que el tamaño actual del programa se hallará más probablemente.

La fórmula del intervalo de predicción es:

$$Rango = t(\alpha/2, n-2)\sigma \sqrt{1 + \frac{1}{n} + \frac{(x_k - x_{prom})^2}{\sum_{i=1}^{n}(x_i - x_{prom})^2}}$$

X_i: número de LOC Objeto estimadas para cada programa en la base de datos histórica.
X_{prom}: es el promedio de LOC Objeto estimadas para cada programa en la base de datos histórica.
n: cantidad de programas almacenados en la base de datos histórica.
X_k: LOC Objetos estimadas en el nuevo programa.
α/2: % utilizado para el intervalo de predicción.

σ: desviación estándar de los datos alrededor de la línea de regresión.

$$Varianza = \sigma^2 = \left(\frac{1}{n-2}\right)\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 * x_i)^2$$

$$DesvEst = \sqrt{Varianza} = \sigma$$

2.5.7. Significado del intervalo de predicción

La calidad de una estimación hecha con el método PROBE es directamente proporcional a la calidad de los datos utilizados. También depende del grado en que estos datos se corresponden a la manera en que se intentará desarrollar el próximo programa. Si la base de datos histórica tiene variaciones bruscas, descontroladas; el intervalo de predicción será grande.

Si se cambia el método de diseño, se construye un programa mucho mayor, o se desarrolla una nueva clase de aplicación, los datos históricos no representaran con precisión lo que se intenta hacer.

Se tiene que reconocer que el intervalo de predicción no es un buen indicador del rango de error de la estimación realizada.

2.5.8. Categorías de tamaños de objetos

Las categorías de tamaños de los objetos son necesarias para establecer las bases de estimación de los nuevos objetos que componen los productos planificados.

Nótese en la Tabla 2.5.8. que el rango de tamaño de los objetos es de 18 a 558 LOC, una proporción de alrededor de 30 a 1.

Debido a que existe un interés en la medida del tamaño relativo de los objetos basado en la consideración de su complejidad funcional, es útil normalizar el tamaño de los objetos, dividiendo la cantidad total de LOC de los objetos entre el número de métodos de cada objeto.

De esta manera, si se tiene un objeto complejo con un único método, se podrá distinguir claramente de un objeto simple con muchos métodos.

Mientras que esto representa una proporción de aproximadamente 10 a 1, es algo más útil que la proporción de 30 a 1, que se tiene sin normalizar.

Los rangos de tamaños son más útiles si son razonablemente pequeños.
Para estimar el tamaño relativo de los objetos se utiliza la desviación estándar.

Nombre del Objeto	Número de Métodos	LOC del Objeto	LOC por Método
each_line	3	31	10,333
each_char	3	18	6,000
list_clump	4	87	21,750
character	3	87	29,000
single_character	3	25	8,333
string_read	3	18	6,000
list_clp	4	89	22,250
char	3	85	28,333
single_char	3	37	12,333
converter	10	558	55,800
string_manager	4	82	20,500
string_builder	5	82	16,400
string_decrementer	10	230	23,000

Tabla 2.5.8. Tabla de tamaño de objetos. El tamaño de los objetos se normaliza dividiendo la cantidad de LOC entre el número de métodos.

$$Varianza = \sigma^2 = (1/n)\sum_{i=1}^{n}\left(x_i - x_{prom}\right)^2$$

$$DesvEst = \sqrt{Varianza} = \sigma$$

, X_i es la cantidad de LOC por método, y X_{prom} es el promedio de las X_i .

DesvEST es una abreviatura utilizada para denotar la desviación estándar.
Desafortunadamente, debido a que los datos de tamaño no están distribuidos normalmente, el cálculo del tamaño de los objetos es un problema más

complicado que la simple medición de la varianza por encima y por debajo de la media.

A continuación se muestra un ejemplo de cómo obtener los rangos de tamaño, y seguidamente se explica como solucionar el problema de la distribución de los datos.

Nombre del Objeto	LOC por Método	$(LOC-LOC_{prom})^2$
each_line	10,333	93,494
each_char	6,000	196,072
list_clump	21,750	3,054
character	29,000	80,954
single_character	8,333	136,171
string_read	6,000	196,072
list_clp	22,250	5,051
char	28,333	69,402
single_char	12,333	58,817
converter	55,800	1281,456
string_manager	20,500	0,247
string_builder	16,400	12,978
string_decrementer	23,000	8,985
Total	260,033	2142,753
Promedio	20,003	
Varianza = Total / n = σ^2		164,827
Desviación Estándar = $\sqrt{Varianza} = \sigma$		12,839

Tabla 2.5.8.1. Cálculo de la Varianza y de la Desviación Estándar. Datos de tamaño no distribuidos normalmente.

2.5.9. Obtención de los rangos de tamaño

Los cinco rangos de tamaño son[9]: Muy Pequeño, Pequeño, Medio, Grande, y Muy Grande.

[9] Consultar el método difuso de Putnam para comprender el significado de los rangos de tamaño, véase la sección 2.4.

La desviación estándar para el tamaño de los 13 objetos en la Tabla 2.5.8.1. es de 12. 839 LDC, lo cual significa que las fronteras de los rangos de tamaño son:

Muy Pequeño (MP) = -5.68

Pequeño (P) = 7.16

Medio (M) = 20.0

Grande (G) = 32.84

Muy Grande (MG) = 45.68

Rango de Tamaño	Fronteras en LOC	LOC por Método
MP	-5,68	
		6
		6
P	7,16	
		8,333
		10,333
		12,333
		16,400
M	20,0	
		20,500
		21,750
		22,250
		23,000
		28,333
		29,000
G	32,84	
MG	45,68	
		55,800

Tabla 2.5.9. Tamaños de objetos organizados en rangos.

Los pasos consisten en calcular el promedio de LOC por métodos para cada objeto, se hace corresponder este valor con la frontera del rango de tamaño Medio, y luego se le resta y adiciona sucesivamente el valor de la desviación estándar, para obtener las valores por debajo y por encima de la frontera Media respectivamente.

2.5.10. Ajuste de la distribución

Un truco para manipular el problema de la distribución no normal de los datos de tamaño, es calcular el logaritmo natural de los datos, computar la desviación estándar de los datos logarítmicos, y luego aplicar antilogaritmos (la función exponencial) para restablecer los datos a su forma original. La Tabla 2.4.10. muestra la aplicación de este procedimiento.

Nombre del Objeto	LOC por Método	*ln(LOC por Método)*	$(\ln LOC - \ln LOC_{prom})^2$
each_line	10,333	2,335	0,2173
each_char	6,000	1,792	1,0196
list_clump	21,750	3,080	0,0773
character	29,000	3,367	0,3201
single_character	8,333	2,120	0,4641
string_read	6,000	1,792	1,0196
list_clp	22,250	3,102	0,0905
char	28,333	3,344	0,2943
single_char	12,333	2,512	0,0836
converter	55,800	4,022	1,4890
string_manager	20,500	3,020	0,0479
string_builder	16,400	2,797	0,0000
string_decrementer	23,000	3,135	0,1115
Total	260,033	36,419	5,2348
Promedio	20,003	2,802	
Varianza	164,827		0,4027
Desviación Estándar	12,839		0,6346
ln MG = 2,802 + 1,269	4,071	MG = $e^{4,071}$ =	58,62
ln G = 2,802 + 0,635	3,437	G = $e^{3,437}$ =	31,09
ln M = 2,802	2,802	M = $e^{2,802}$ =	16,48
ln P = 2,802 - 0,635	2,167	P = $e^{2,167}$ =	8,73
ln MP = 2,802 - 1,269	1,533	MP = $e^{1,533}$ =	4,63

Tabla 2.5.10. Ejemplo de ajuste de la distribución.

Adaptado de(*Estimating of Software Size - Part II* 2003).

2.6. Estimación en Proyectos de Desarrollo Incremental

Un enfoque para la gestión de proyectos de desarrollo incremental utiliza la información del tamaño del software para cuantificar el riesgo en la estimación del tiempo y el esfuerzo de desarrollo del producto.

En este contexto, las métricas utilizadas para estimar el tamaño del software pueden ser cualquiera de una amplia variedad de medidas, tales como NLOC[10], sentencias de código fuente (SS), componentes, objetos o puntos de función (PF) .

En el enfoque utilizado se especifica el rango de tamaño estimado para cada subsistema o módulo a desarrollar. Las entradas de rango de tamaño para un subsistema dado son expresadas como se muestra en la Tabla 2.6.

		Software que se reutiliza/Paquete de software		Software nuevo/modificado		
Nombre/Número del Subsistema	Lenguaje de programación de SS/**NLOC**	Tamaño original **NLOC**/FP	% a ser modificado	Rango de tamaño estimado en SS/**NLOC**/PF		
				Mín.	Más probable /Medio	Máx.
1	C++	5 000	30	6000	6 250	6500
2	Java	9 000	45	9000	9250	9500
3	Object Pascal	6 000	25	6250	7125	8000
4	C#	8 000	40	8100	8150	8200

Tabla 2.6. Rangos de tamaño de subsistemas.

[10] NLOC: Líneas de código efectivas. Para más información véase Capítulo 1, sección 1.2.1.

El **software que se reutiliza** identifica al software existente que será modificado y extendido. La funcionalidad del subsistema puede ser implementada a partir de software ya disponible.

El tamaño de un subsistema es expresado a partir del rango de tamaño especificado[11]. Cuando se cuenta con información detallada de un subsistema puede esperarse que el rango de tamaño asociado sea relativamente estrecho. En caso de que las especificaciones de un subsistema sean escasas en algunas áreas, el rango de tamaño puede esperarse que sea amplio.

La incertidumbre en la estimación del tamaño es utilizada para cuantificar el riesgo en la estimación del tiempo de desarrollo, esfuerzo, costo y fiabilidad. Igualmente la estimación del tamaño y sus rangos imprecisos proveen los medios para rastrear el progreso del proyecto a medida que el software es desarrollado. Se recomienda subdividir los subsistemas de tal forma que sus tamaños medios no excedan las 3000 NLOC.

2.6.1. Dependencia de incremento de un subsistema

Cuando se planifica un desarrollo iterativo o incremental entonces la Tabla 2.6.1. se utiliza para identificar cuales subsistemas están siendo realizados en un determinado incremento. El por ciento estimado del código de un subsistema para cada iteración, permite identificar:

- incrementos que consisten en una colección de subsistemas enteros, en este caso cada subsistema se cuantifica como un 100%

- incrementos que consisten en partes de un subsistema, en este caso cada subsistema se estima como un por ciento del tamaño total del subsistema

[11] En este caso debe establecerse un procedimiento para estimar los valores de rangos de tamaño. Un enfoque compatible podría ser el método difuso de Putnam, véase la sección 2.4.

Número del Subsistema	Iteración/Número del incremento – Por ciento del Tamaño del Subsistema						Rango de Tamaño del Subsistema Total		
	1:%	2:%	3:%	4:%	5:%	6:%	Mín.	Medio	Máx.
1									
2									
3									

Tabla 2.6.1. Registro de incrementos de subsistemas.

Donde los incrementos son identificados como consistentes de un número de subsistemas parciales, puede ser evidencia de que el primer incremento de código de un subsistema se espera revisar y extender para incrementos posteriores. Esto implica a su vez, que el código inicial puede probablemente ser cambiado, como incrementos posteriores son desarrollados, y por esta razón puede esperarse un nuevo procesamiento para el número creciente de incrementos más tempranos y los subsistemas cambiados.

2.6.2. Medición de los incrementos

A partir de la información mostrada en la Tabla 2.6.1. es posible obtener el rango de tamaño total para cada subsistema. La Tabla 2.6.2. expone la forma simple de capturar el total de rango de tamaño de cada incremento.

Esto entonces se convierte en la base para un seguimiento de alto nivel del progreso en el desarrollo de un incremento dado. La fecha de inicio y de terminación del incremento, más su tamaño, pueden ser usados para monitorear el progreso, incluso a nivel global del proyecto.

Número del Incremento	Fechas		Rango de Tamaño		
	Inicio	Fin	Mín.	Más probable	Máx.
1					
2					
3					

4					
5					

Tabla 2.6.2. Plan de Incrementos y Rangos de Tamaño.

2.6.3. Una muestra de seguimiento mediante incrementos

Para ilustrar el seguimiento mediante incrementos, la Figura 2.6.3 muestra un ejemplo en el que se da seguimiento al progreso de un desarrollo a nivel de incremento utilizando una herramienta automática, SLIM-Control.

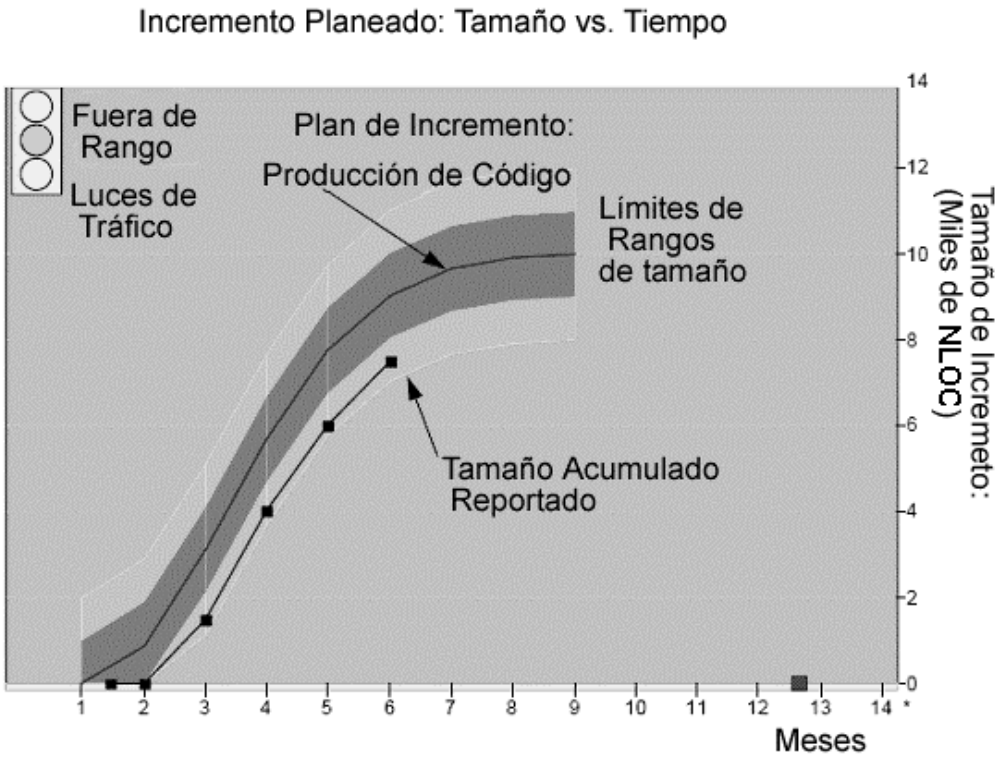

Figura 2.6.3. Rastreo de un incremento.

Adaptado de(GREENE).

2.7. Descomposición funcional del problema

Para la descomposición funcional del problema se aplica la estrategia **divide y vencerás.** Un problema complejo se parte en problemas más pequeños que resultan más manejables. Esta es la estrategia que se aplica al inicio de la planificación del proyecto.

Las funciones del software, descritas en la exposición del ámbito, se evalúan y refinan para proporcionar más detalle antes del comienzo de la estimación.

2.7.1. Estimación basada en la descomposición funcional del problema

El planificador del proyecto comienza con un enfoque limitado para el ámbito del software y desde este estado intenta descomponer el software en funciones que se pueden estimar individualmente. Para cada función entonces se estiman las LDC[12] y el PF (la variable de estimación). De forma alternativa, el planificador puede seleccionar otro componente para dimensionar clases u objetos, cambios o procesos de gestión en los que puede tener impacto.

Para estimaciones de LDC, la descomposición se centra en las funciones del software.

Las técnicas de estimación de LDC y PF difieren en el nivel de detalle que se requiere para la descomposición y el objetivo de la partición. Cuando se utiliza LDC como variable de estimación, la descomposición[13] es absolutamente esencial y con frecuencia se toman para considerables niveles de detalle.

[12] Esta sección se refiere a las líneas de código como LDC, véase (PRESSMAN 2002).
[13] Para la descomposición también puede usarse una lista de componentes estándar.

A continuación se muestra una estrategia de descomposición(PHILLIPS 1998):

definir el ámbito del producto;
identificar funciones descomponiendo el ámbito;
hacer mientras haya funciones;
seleccionar una [función $_j$];
asignar todas las funciones a la lista de subfunciones;
hacer mientras haya subfunciones;
seleccionar una [subfunción $_k$] ;
si [subfunción $_k$] = [subfunción $_d$] descrita en una base de datos histórica;
entonces
anotar datos históricos del coste, esfuerzo, tamaño (LDC o PF) para
la [subfunción $_d$];
ajustar datos históricos del coste, esfuerzo, tamaño basados en
cualquier diferencia;
use datos del coste, esfuerzo, tamaño ajustados para obtener una
estimación parcial, Ep;
estimación proyecto = suma de {Ep} ;
si no
si se puede estimar coste, esfuerzo, tamaño (LDC o PF)
para [subfunción $_k$]
entonces
obtener estimación parcial Ep;
estimación proyecto = suma de {Ep};
si no
subdividir [subfunción $_k$] en subfunciones mas pequeñas;
añadirlas a la lista de subfunciones;
fin si
fin si
fin hacer
fin hacer

El enfoque de descomposición visto anteriormente asume que todas las funciones pueden descomponerse en subfunciones que podrían asemejarse a

entradas de una base de datos histórica. Si este no es el caso, deberá aplicarse otro enfoque de dimensionamiento.

Cuanto más grande sea el grado de particionamiento, más probable será que puedan desarrollarse estimaciones de LDC más exactas.

Para estimaciones de PF, la descomposición funciona de diferente manera. En lugar de centrarse en la función, se estiman cada una de las características del dominio de información [14]. Las estimaciones resultantes se pueden utilizar para derivar un valor de PF que se pueda unir a datos pasados y utilizar para generar una estimación.

Con independencia de la variable de estimación que se utilice, el planificador del proyecto comienza por estimar un rango de valores para cada función o valor del dominio de información. Mediante los datos históricos o (cuando todo lo demás falla) la intuición, el planificador estima un valor de tamaño optimista, más probable y pesimista para cada función, o cuenta para cada valor del dominio de información. Al especificar un rango de valores, se proporciona una indicación implícita del grado de incertidumbre.

Adaptado de (PRESSMAN 2002).

2.8. Metodología para el conteo de los Puntos de Función

A continuación se describen algunas de las metodologías más importantes para el cálculo de los Puntos de Función. Para mayores detalles se recomiendo consultar los manuales oficiales de estas metodologías. Véase por ejemplo (*IFPUG*).

[14] El término *función o valor del dominio de información* se hace corresponder con el término *funcionalidades* utilizado en la sección 2.8.

2.8.1. IFPUG-FPA (Function Point Analisys)

La técnica de medición del tamaño en Puntos de Función (PF) consiste en asignar una cantidad de "puntos" a una aplicación informática según la complejidad de los datos que maneja y de los procesos que realiza sobre ellos. Siempre tratando de considerarlo desde el punto de vista del usuario.

El método IFPUG-FPA establece los siguientes pasos:

- Determinar el tipo de cálculo a realizar.
- Identificar las fronteras del sistema.
- Contar las funciones tipo datos.
- Contar las funciones tipo transacciones.
- Calcular los Puntos Funcionales[15] sin ajustar.
- Determinar los modificadores y el ajuste.
- Calcular los puntos funcionales ajustados.

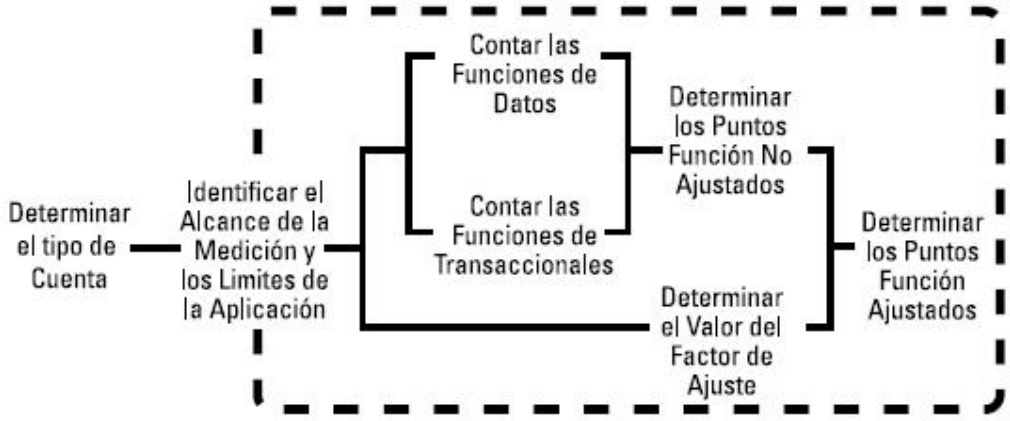

Figura 2.8. Diagrama del método IFPUG –FPA. Adaptado de(DURÁN 2005).

[15] Esta sección se refiere a los *Puntos de Función* como *Puntos Funcionales*, término utilizado en el documento original (GROMPONE 1996).

2.8.1.1. Determinar el tipo de cálculo a realizar

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

- Proyecto nuevo a desarrollar, o en desarrollo.
- Ampliación de un proyecto existente y completado.
- Replanteo de un proyecto existente y completado.

Las tres situaciones poseen interés. El caso de un proyecto nuevo es la situación típica. La ampliación de un proyecto que se hizo aplicando la metodología es, sin duda, el caso más simple de todos: se trabaja en un terreno conocido y se tienen cifras previas.

El replanteo de un proyecto ya completado tiene una importancia muy especial en los casos de emigración de un sistema viejo a una nueva tecnología.

2.8.1.2. Identificar las fronteras del sistema

La determinación de las fronteras del sistema en estudio es el primer paso del análisis. La cantidad de puntos funcionales de un sistema depende de una manera no trivial, de la elección de las fronteras. Por esta razón se debe ser cuidadoso en este punto.

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

Cuando se amplia un proyecto existente, las fronteras se deben considerar en forma coherente. De otra forma no es posible aplicar las ecuaciones y la metodología. En caso que la ampliación exija una redefinición de las fronteras,

se debe replantear el proyecto existente de acuerdo con la nueva definición de las fronteras.

2.8.1.3. Contar las funciones tipo datos

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

Existen dos grandes grupos:

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

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

Para cada uno de los archivos es necesario determinar su complejidad. La complejidad depende de la organización lógica de los datos considerados. Se consideran dos tipos de conceptos:

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

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

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

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

Complejidad de los archivos lógicos			
	de 1 a 19 DET	de 20 a 50 DET	más de 50 DET
sólo 1 RET	simple	simple	promedio
de 2 a 5 RET	simple	promedio	complejo
más de 5 RET	promedio	complejo	complejo

2.8.1.4. Contar las funciones tipo transacciones

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

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

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

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

Para cada uno de los archivos es necesario determinar su complejidad. La complejidad depende de la organización lógica de los datos considerados […] Se consideran dos tipos de conceptos:

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

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

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

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

Complejidad de las entradas externas			
	de 1 a 4 DET	de 5 a 15 DET	más de 15 DET
hasta 1 FTR	simple	simple	promedio
2 FTR	simple	promedio	complejo
más de 2 FTR	promedio	complejo	complejo

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

Complejidad de las salidas externas			
	de 1 a 5 DET	de 6 a 19 DET	más de 19 DET
hasta 1 FTR	simple	simple	promedio
2 a 3 FTR	simple	promedio	complejo
más de 3 FTR	promedio	complejo	complejo

La manera de determinar la complejidad de las consultas[16] es consecuencia del hecho de tratarse de una operación doble. A estos efectos se procede como se indica:

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

2.8.1.5. Cálculo de los Puntos Funcionales sin ajustar

Los Puntos Funcionales sin ajustar se calculan completando la Tabla 2.8.1.

Funcionalidades	Complejidad	Peso	Cantidad	Total = Peso * Cantidad
Archivos lógicos	simple	7		
	promedio	10		
	complejo	15		
Interfaces externas	simple	5		
	promedio	7		
	complejo	10		
Entradas externas	simple	3		
	promedio	4		
	complejo	6		
Salidas externas	simple	4		
	promedio	5		
	complejo	7		
Consultas externas	simple	3		
	promedio	4		
	complejo	6		
Puntos Funcionales sin ajustar(suma de totales) =				

Tabla 2.8.1. Tabla para el cálculo de los Puntos Funcionales sin ajustar(*Instructivo para la Cuenta de Puntos Función*).

[16] Este documento adopta una forma bastante lógica de determinar la complejidad de las consultas. No obstante, fuentes más actualizadas utilizan el mismo criterio que para el cálculo de la complejidad de las salidas.

2.8.1.6. Las características modificadoras

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

Cada una de las características debe ser evaluada mediante una escala de 0 a 5, véase la Tabla 2.8.1.1. A los efectos de la normalización, existen definiciones que permiten ajustar la evaluación a una práctica precisa(RENDÓN 2007).

0	No influencia	Ninguna	0%	0 – 10%
1	Incidental	Insignificante	1 - 20%	11 – 20%
2	Moderado	Moderada	21 - 40%	21 – 30%
3	Medio	Media	41 – 60%	31 – 40%
4	Significativo	Significativa	61 – 80%	41 – 50%
5	Esencial	Fuerte	81 – 100%	> 50%

Tabla 2.8.1.1. Escala para la evaluación de las características modificadoras. Por decisión del equipo de trabajo se pueden asumir valoraciones en porcentajes, como se muestra(RENDÓN 2007).

Las características modificadoras son(PRESSMAN 2002):

1. ¿Requiere el sistema copias de seguridad y de recuperación fiables?
2. ¿Se requiere comunicación de datos?
3. ¿Existen funciones de procesamiento distribuido?
4. ¿Es crítico el rendimiento?
5. ¿Se ejecutará el sistema en un entorno operativo existente y fuertemente utilizado?
6. ¿Requiere el sistema entrada de datos interactiva?
7. ¿Requiere la entrada de datos interactiva que las transacciones de entrada se lleven a cabo sobre múltiples pantallas u operaciones?

8. ¿Se actualizan los archivos maestros de forma interactiva?
9. ¿Son complejas las entradas, las salidas, los archivos o las peticiones?
10. ¿Es complejo el procesamiento interno?
11. ¿Se ha diseñado el código para ser reutilizable?
12. ¿Está incluidas en el diseño la conversión y la instalación?
13. ¿Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones?
14. ¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?

2.8.1.7. Cálculo del factor de ajuste

El factor de ajuste se calcula a partir de los valores asignados a las características modificadoras (CM). Cada punto asignado agrega 1% al factor de ajuste (FA) mediante la ecuación:

$$FA = 0.65 + 0.01 * \sum CM$$

$\sum$ CM: sumatoria de los 14 valores asignados a las características modificadoras.

2.8.1.8. Cálculo de los Puntos Funcionales ajustados

Luego de haber calculado los Puntos Funcionales sin ajustar (PFSA), y el factor de ajuste (FA), se procede al cálculo de los Puntos Funcionales ajustados (PFA). Se aplica la siguiente ecuación:

$$PFA = PFA * FA$$

Adaptado de(GROMPONE 1996).

2.8.2. Puntos Característica (Feature Points)

Los Puntos Característica son una adaptación de la métrica de Puntos de Función a sistemas software científicos y de ingeniería.

A diferencia de los sistemas de gestión, este otro tipo de sistemas tiene dos características básicas:

1. La complejidad algorítmica que implementan.
2. El escaso número de entradas y salidas que suelen tener.

Puesto que el conteo de Puntos Función no contempla para nada la complejidad algorítmica de los sistemas, esto hace que los resultados obtenidos no sean realmente significativos. En general, y para estos sistemas, el resultado obtenido tras un Análisis de Puntos de Función suele ser bastante inferior (entre un 20 y un 35%) al que se obtiene con otras técnicas que sí contemplan la complejidad algorítmica.

Un algoritmo se define como un problema de complejidad computacional limitada que se incluye dentro de un determinado programa de computadora. La inversión de una matriz, la decodificación de una cadena de bits o el manejo de una interrupción son todo ellos ejemplos de algoritmos(GIRALDO).

Para calcular los puntos de características, nuevamente se cuentan los valores del ámbito de información de forma análoga a IFPUG-FPA. Luego se completa la Tabla 2.8.2.

Funcionalidades	Cantidad	Peso	Total = Cantidad* Peso
Archivos		7	
Interfaces		7	
Entradas		4	
Salidas		5	
Consultas		4	
Algoritmos		3	
Puntos de Características sin ajustar (suma de totales) =			

Tabla 2.8.2. Tabla para el cálculo de los Puntos Característica. Adaptación de la tabla mostrada en(GIRALDO).

Se usa un único valor de peso para cada uno de los parámetros de medida y se calcula el valor del punto característica global mediante la ecuación:

PC = PCSA * FA

PC: Puntos de Características globales, o ajustado.

PCSA: Puntos de Características sin ajustar.

FA: factor de ajuste[17].

Debe tenerse en cuenta que los puntos de característica y los puntos de función representan lo mismo, "funcionalidad o utilidad" en forma de software.

Adaptado de(V. 2006).

2.8.3. MK II

El método MK-II es una derivación del de Puntos de Función de Albrecht[18], y tiene cómo principal área de aplicación los Sistemas de Información. El método

[17] El factor de ajuste se determina como en el IFPUG-FPA, véase sección 2.8.1.7.
[18] Al método original de cálculo de los Puntos de Función se le suele llamar *método de Albrecht*, en honor a su creador.

MK-II especifica, además, un procedimiento para el cálculo del esfuerzo (basado en una fórmula de productividad) y de la duración.

El cálculo de la complejidad en MK-II mantiene la misma estructura que [IFPUG-FPA], variando la forma de cálculo de los puntos de función sin ajustar y cambiando (y añadiendo) los factores de complejidad técnica[19] (19 en MK-II).

MK-II considera que un sistema de información se compone de transacciones lógicas, siendo cada transacción lógica un conjunto de entrada, proceso y salida. Cada transacción tendrá un número determinado de tipos de datos de entrada, tipos de datos de salida y entidades de datos afectadas.

La fórmula de MK-II para una transacción lógica, siendo el total es la suma de los NPFSA - Número de Puntos de Función Sin Ajustar - de todas las transacciones lógicas, es la siguiente:

NPFSA = PI x (nº de tipos de elementos de datos de entrada) + PE x (nº de tipos de entidad primaria referenciadas por la transacción) + PO x (nº de tipos de elementos de datos de salida)

PI: peso de las entradas.
PE: peso de las entidades.
PO: peso de las salidas.

PI, PE y PO son parámetros que se obtienen por calibración del modelo con una base histórica de proyectos.

Los valores de éstos parámetros se expresan a continuación:

PI	PE	PO
0,58	1,66	0,26

[19] La denominación *factores de complejidad técnica* es equivalente a *características modificadoras*. Véase la sección 2.8.1.6.

Una característica de estos parámetros es que por convenio debe cumplirse que:

PI + PE + PO = 2,5

Por cada parte del sistema software con la misma complejidad técnica se corrigen los puntos de función sin ajustar (NPFSA) para obtener los puntos de función ajustados (NPF) mediante la fórmula:

NPF = FCT x NPFSA

El Factor de Complejidad Técnica (FCT) se obtiene a su vez mediante:

FCT = 0,65 + C x ∑GI

Dónde C es un parámetro medio de la industria (= 0,005) y ∑GI es el sumatoria de los grados de influencia de cada factor (1...19) que puede tomar valor entre 0 y 5. Los elementos que tiene en cuenta MK-II son:

1. Comunicaciones.
2. Funciones distribuidas.
3. Rendimiento.
4. Configuración de alto uso.
5. Tasas de Transacciones.
6. Entrada en línea.
7. Diseño para eficiencia de usuario final.
8. Actualización en línea.
9. Complejidad del proceso.
10. Utilizable en otras aplicaciones.
11. Facilidades de Instalación.
12. Facilidades de Operación.
13. Emplazamientos múltiples.
14. Facilidad de cambio.
15. Requisitos de otras aplicaciones.

16. Seguridad, Privacidad, Auditabilidad.
17. Necesidades de formación de usuarios.
18. Uso directo por terceras partes.
19. Documentación.

La metodología MK-II añade nuevos factores de complejidad técnica a los listados por el IFPUG-FPA. Estos nuevos factores tienen sentido, pero solamente interesan en proyectos grandes, de cierto(GROMPONE 1996).

Adaptado de(INFORMÁTICA 1999).

2.9. Estimaciones Combinadas

Un buen método para mejorar la precisión de las estimaciones es hacienda estimaciones por varias vías, y luego calcular un promedio ponderado de las estimaciones. Por ejemplo, para promediar cuatro estimaciones obtenidas por diferentes técnicas:

1. Asignar un peso a cada estimación de tal manera que la suma de todos los pesos sea 1. Los pesos más pequeños indican confianza en la estimación mientras que los pesos más grandes indican incertidumbre.
2. Computar el tamaño ponderado para cada estimación (Tamaño * Peso).
3. Sumar los tamaños ponderados para obtener la estimación combinada.

La Tabla 2.9 ilustra el cálculo de una estimación combinada del tamaño de 4 estimaciones con un resultado de 129 KLOC.

KLOC	Peso	Tamaño Ponderado
120	0,30	36
160	0,40	64
100	0,20	20
90	0,10	9
Total =		129

Tabla 2.9. Cálculo de una estimación combinada del tamaño.

Adaptado de(Software Estimation).

2.9.2. Caso particular

Un caso particular de este método, muy utilizado en las estimaciones del software, es cuando sólo se identifican tres valores (estimación de tres puntos): un valor optimista del tamaño (el más pequeño), un valor pesimista (el más grande), y el valor más probable, este último es con frecuencia el promedio de los tamaños estimados.

2.10. Métricas y Estimación en Proyectos Orientados a Objetos

Las líneas de código no se aplican a la estimación del Software Orientado a Objetos (OO) por cuanto se persigue la reutilización, en su lugar se utilizan otras medidas más apropiadas como son las clases y otras métricas relacionadas. Los Puntos de Función pueden ser utilizados pero no provee una granularidad suficiente para ajustes de planificación y esfuerzo a realizar, los cuales se requieren cuando se itera a través del paradigma recursivo/paralelo(PRESSMAN 2002).

Lorenz y Kidd sugieren el siguiente conjunto de métricas para proyectos OO (LORENZ 1994):

Número de guiones de escenario. Un guión de escenario es una secuencia detallada de pasos que describen la interacción entre el usuario y la aplicación. Cada guión se organiza en tripletes de la forma:

{iniciador, *acción,* **participante}**

donde **iniciador** es el objeto que solicita algún servicio (que inicia un mensaje); *acción* es el resultado de la solicitud; y **participante** es el objeto servidor que

cumple la petición (solicitud). El número de guiones de actuación esta directamente relacionada al tamaño de la aplicación.

Número de clases clave. Las clases clave son las componentes altamente independientes definidas inicialmente en el Análisis OO (AOO). Debido a que estas clases son centrales al dominio del problema, el número de dichas clases es una indicación del esfuerzo necesario para desarrollar el software y de la cantidad potencial de reutilización a aplicar durante el desarrollo del sistema.

Número de clases de soporte. Las clases de soporte son necesarias para implementar el sistema, pero no están directamente relacionadas con el dominio del problema. Como ejemplos podemos tomar las clases de Interfaz Grafica de Usuario, el acceso a bases de datos y su manipulación y las clases de comunicación. Además las clases de soporte se definen iterativamente a lo largo del *proceso recursivo/paralelo*.

El número de clases de soporte es un indicador del esfuerzo necesario requerido para desarrollar el software y de la cantidad potencial de reutilización a aplicar durante el desarrollo del sistema.

Número promedio de clases de soporte por clase clave. En general las clases clave son conocidas en las primeras etapas del proyecto. Las clases de soporte se definen a lo largo de este. Si, dado un dominio de problema, se conociera la cantidad promedio de clases de soporte por clase clave, la estimación (basada en la cantidad total de clases) se simplificaría mucho.

Lorenz y Kidd sugieren que las aplicaciones con IGU poseen entre dos y tres veces más clases de soporte que clases clave. Las aplicaciones sin IGU poseen a lo sumo dos veces más de soporte que clave.

Número de subsistemas. Un subsistema es una agregación de clases que dan soporte a una función visible al usuario final del sistema. Una vez que se identifican los subsistemas, resulta más fácil realizar una planificación

razonable en la cual el trabajo en los subsistemas está repartido entre los miembros del proyecto.

2.10.1. Un enfoque OO para estimaciones y planificación

Se propone el siguiente enfoque para la estimación(LORENZ 1994):

1. Desarrollo de estimaciones usando un método aplicable a aplicaciones convencionales.

2. Desarrollar guiones de escenario y determinar una cuenta, usando Análisis OO (AOO). Reconocer que el número de guiones de escenarios puede cambiar con el progreso del proyecto. El número de guiones de escenario es proporcional al tamaño del software a desarrollar.

3. Determinar la cantidad de clases clave usando AOO.

4. Clasificar el tipo de interfaz para la aplicación y desarrollar un multiplicador para las clases de soporte:

Tipo de Interfaz	Multiplicador
Interfaz no gráfica (No IGU)	2,0
Interfaz de usuario basada en texto	2,25
Interfaz Gráfica de Usario (IGU)	2,5
Interfaz Gráfica de Usuario (IGU) compleja	3,0

Multiplicar el número de clases clave (paso 3) por el multiplicador anterior para obtener una estimación del número de clases de soporte.

5. Relacionar la cantidad total de clases (clave + soporte) con otros datos de tamaño para derivar una estimación del tamaño del producto a desarrollar.

Un trabajo de la Universidad de Trento, Italia, (MARCO RONCHETTI 2004)encontró una alta correlación entre el número de método de los objetos del análisis y el tamaño del producto de software. Lo anterior se realizó analizando los datos producidos por una organización de software CMM nivel 3. Se utilizaron un conjunto variado de métricas OO.

Adaptado de(PRESSMAN 2002).

CAPITULO 3: PROPUESTA DEL MODELO DE SOLUCION

3.1. Vista Estructural del Software

Teniendo como base todo el contenido teórico de los capítulos precedentes a continuación se propone un modelo estructural para el producto de software. La intención es facilitar la estimación en base a datos históricos dando una visión del producto que permita gestionar los elementos utilizados para cuantificar el tamaño de la aplicación.

La esencia del modelo se fundamenta en organizar los elementos del software de acuerdo a su pertenencia a elementos de niveles superiores. A cada elemento se asocia la información necesaria para su identificación y categorización, además de los datos de tamaño necesarios utilizados como referencia por el nuevo software que se este desarrollando.

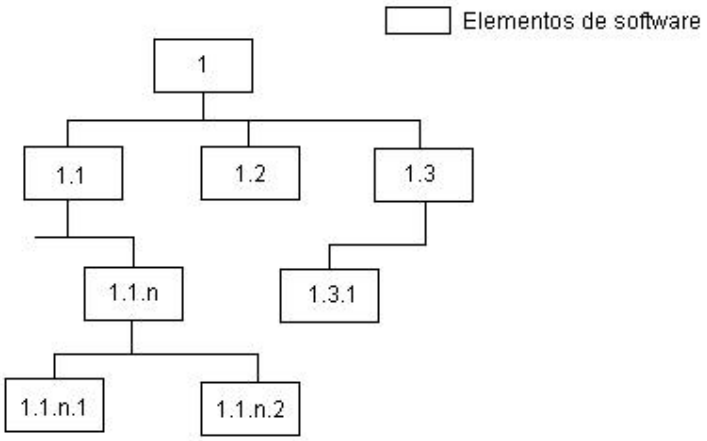

Figura 3.1. Vista estructural del software. Los elementos se organizan de acuerdo a su pertenencia a componentes de niveles superiores.

Para la clasificación de los componentes se puede definir una jerarquía de categorías que agrupe los componentes del software de acuerdo a determinados criterios.

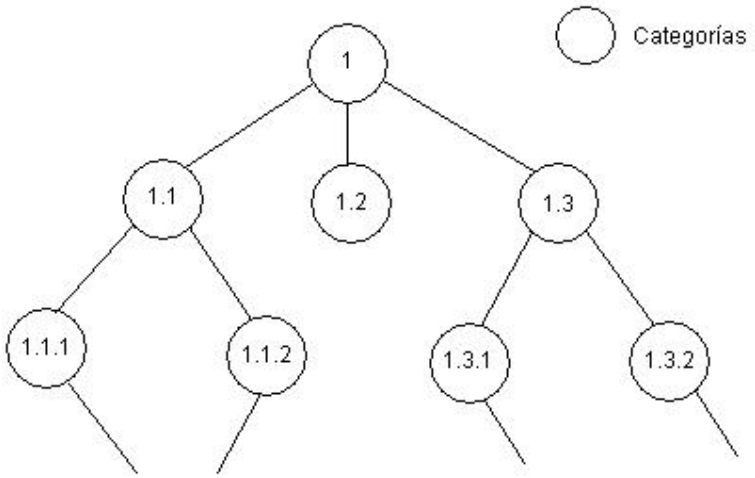

Figura 3.1.1. Jerarquía de categorías para la clasificación de elementos de software.

Puesto que el movernos por el grafo de categorías puede hacerse difícil al incrementarse el número de estas en la base de datos histórica, se recomienda como técnica alternativa la utilización de una lista de descriptores para cada componente en la base de datos. Un descriptor sería una clave, juego de caracteres o palabra, que permite identificar el componente de manera significativa.

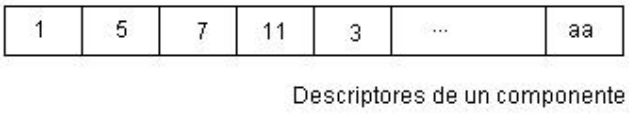

Figura 3.1.2. Vector de descriptores de un componente. La coincidencia ocurre si cada descriptor en el vector *"y"* de búsqueda existe en el vector de descriptores del componente, y además el vector *"exclusion"* no contiene ninguna ocurrencia de los descriptores del componente.

Para la búsqueda se utilizarían tres vectores: el *vector y*, contiene los descriptores que no pueden faltar en la descripción del componente; el *vector o* contiene los descriptores cuya ocurrencia es opcional en la descripción del componente, y el *vector exclusion* contiene los descriptores cuya ocurrencia debe ser nula en la descripción del componente.

De forma alternativa pudiera desarrollarse un intérprete para un lenguaje de consulta que permita la recuperación de los componentes en base a sus descriptores. Es decir, mediante el análisis directo de una expresión cuyos operadores sean *y, o, exclusion*; los operandos serían los descriptores del componente.

Cada elemento del software en la base de datos tiene asociado una medida de su tamaño. Y de forma opcional los datos de métricas que describen al componente.

Un elemento de software podría ser una un subsistema, componente estándar, clases u objetos, procedimiento o función, etc.

La idea es que no se trata de ver una estructura de los elementos de un software acabado, sino ver esta estructura como una herramienta para la descomposición funcional del software en los primeros momentos del ciclo de vida del proyecto.

De esta manera podría analizarse como la estructura evoluciona desde una dimensión funcional (de las funcionalidades requeridas por el software), hasta la concreción de sus partes, en elementos específicos de software, dando soporte a las funcionalidades identificadas.

3.1.1. Definición de clases de programación

A continuación se definen un conjunto de clases basado en los elementos teóricos expuestos:

La clase Entidad. Esta clase soporta la identificación y descripción de cualquier entidad en su contexto. Esto ayuda considerablemente en la documentación de los elementos y componentes que se definan para el producto de software, y en el proceso de aplicación de los métodos de estimación.

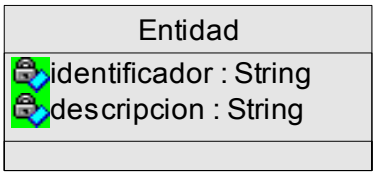

Figura 3.1.3. Diagrama UML de la clase Entidad. El atributo "identificador" permite identificar a la entidad en forma unívoca, mientras que el atributo "descripción" permite describir a la entidad por medio de cualquier comentario.

La clase Categoría. Es una entidad que permita agrupar los elementos del software de acuerdo a múltiples criterios. Esta a su vez se descompone en subcategorías para permitir extender la clasificación, y determinar una jerarquía de tipos que agrupe de forma eficiente y manejable los datos de elementos[20].

El ser una categoría un conjunto de elementos que cumplen determinados criterios permite utilizar las operaciones de conjuntos para operar sobre estas.

Un ejemplo hipotético de intersección de categorías podría ser: obtener los componentes de software que pertenezcan a "Gestión y Optimización".

La clase Elemento. Los elementos se definen como entidades que tienen asociadas varias categorías o descriptores. Estos se descompondrían a su vez en subelementos con características similares y así sucesivamente. A cada elemento se le da la posibilidad de poseer una lista de medidas, y el tamaño del elemento.

[20] En *Ingeniería del software: un enfoque práctico* se aborda la clasificación y recuperación de componentes estándar (PRESSMAN 2002).

Las operaciones básicas incluirían la gestión de los elementos: crear elementos, eliminar elementos, mover elementos, modificar elementos, etc.

Cada elemento delega responsabilidades a sus subelementos, como es el caso de la operación de búsqueda de un elemento, buscarElemento().

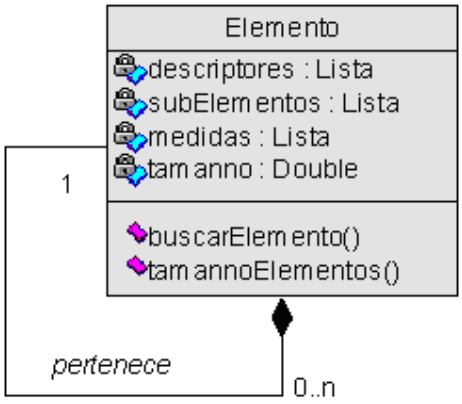

Figura 3.1.4. Diagrama UML de la clase *Elemento*. Las *medidas* son evaluaciones de ciertas características del software que pueden usarse para derivar estimaciones de tamaños.

La función tamannoElementos() permite calcular el tamaño del elemento de software mediante la suma del tamaño de sus subelementos.

Se recomienda consultar el material sobre Ontología de la Medición del Software(MATEUS FERREIRA 2006), donde se exponen los fundamentos teóricos de esta disciplina. Esto podría ayudar a un diseño adecuado de los elementos del software y sus atributos, desde la perspectiva de sus medidas asociadas.

3.2. Automatización del Cálculo de los Puntos de Función

A continuación se presentan los modelos de clases para la cuenta de Puntos de Función.

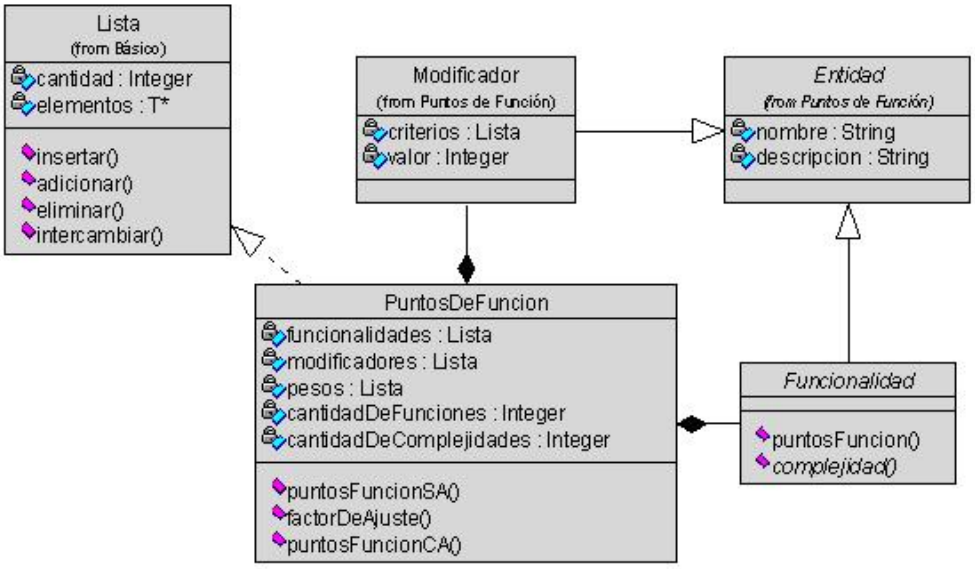

Figura 3.2. Diagrama UML para la clase *PuntosDeFuncion*.

La clase *PuntosDeFuncion* contiene una matriz de los pesos de complejidad de las funcionalidades, una lista de modificadores de la cuenta de puntos de función, y la lista de funcionalidades.

Se definen las operaciones siguientes:

- puntosFuncionSA(): calcula los puntos de función sin ajustar.
- factorDeAjuste(): calcula el factor de ajuste para la cuenta de puntos de función.
- puntosFuncionCA(): calcula el valor de los puntos de función ajustados.

Los modificadores y las funcionalidades constituyen entidades que se documentan.

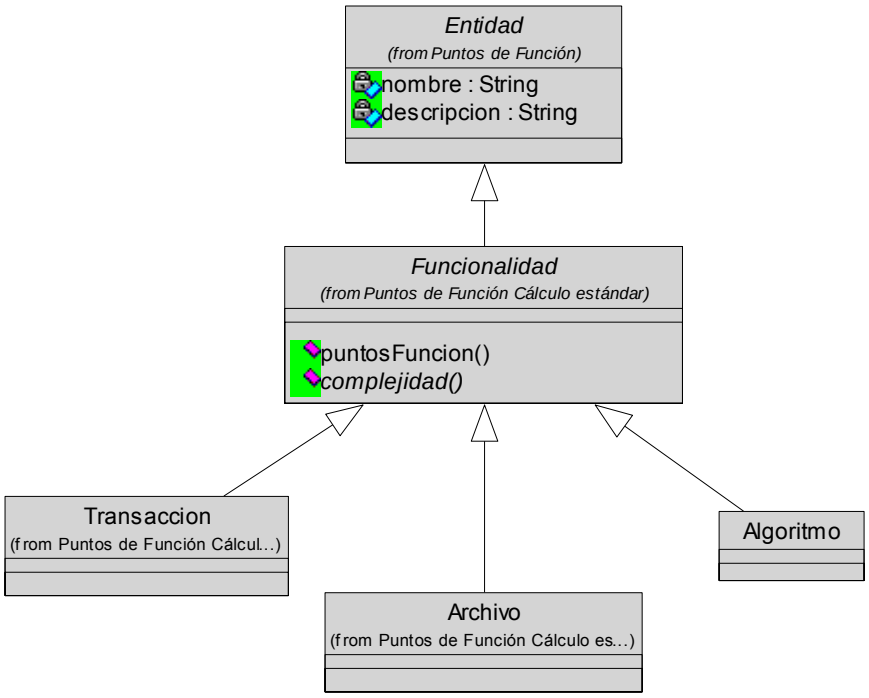

Figura 3.2.1. Diagrama UML para la clase *Funcionalidad*.

La clase *Funcionalidad* define los métodos para el cálculo de la contribución de cada funcionalidad a la cuenta de puntos de función, y de la complejidad asociada.

Constituyen funcionalidades las transacciones, las funcionalidades tipo datos (archivos), y los algoritmos.

Cada funcionalidad es documentada a partir de su identificación y descripción como entidad.

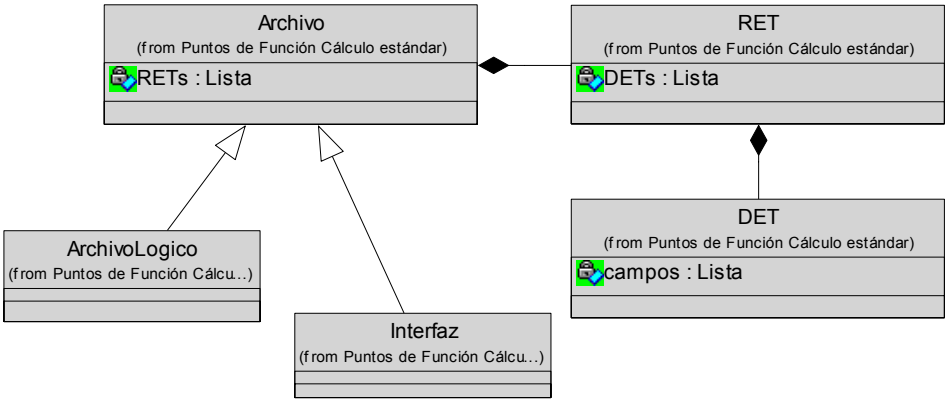

Figura 3.2.2. Diagrama UML para la clase *Archivo*.

Cada archivo (clase *Archivo*) incluye una lista de Tipos de Registro de Datos (RET), y cada RET incluye a su vez una lista de Tipos de Elementos de Datos (DET). Ambos atributos se utilizan para la determinación de la complejidad de los archivos.

La clase *Archivo* incluye las interfaces (clase *Interfaz*) y los archivos lógicos (clase *ArchivoLogico*).

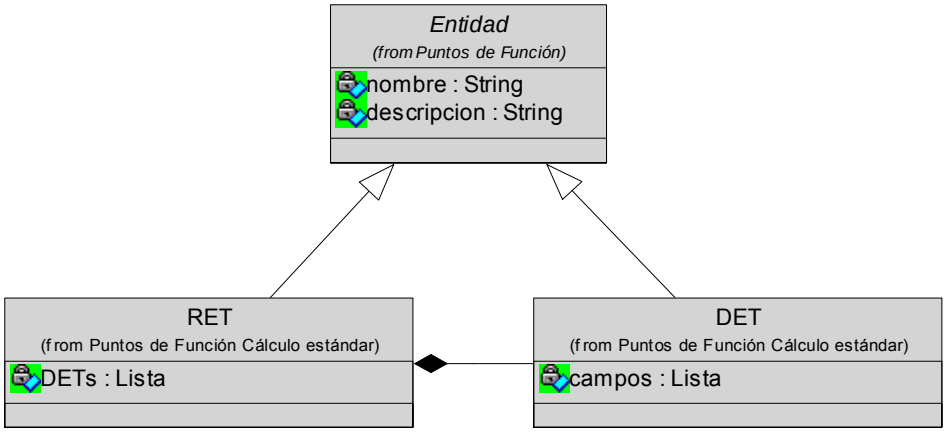

Figura 3.2.3. Diagrama UML para las clases *RET* y *DET*.

La clase *RET* y la clase *DET* heredan de la clase *Entidad* para permitir ser documentadas.

La clase DET incluye una lista de campos relacionados del tipo de datos que representa.

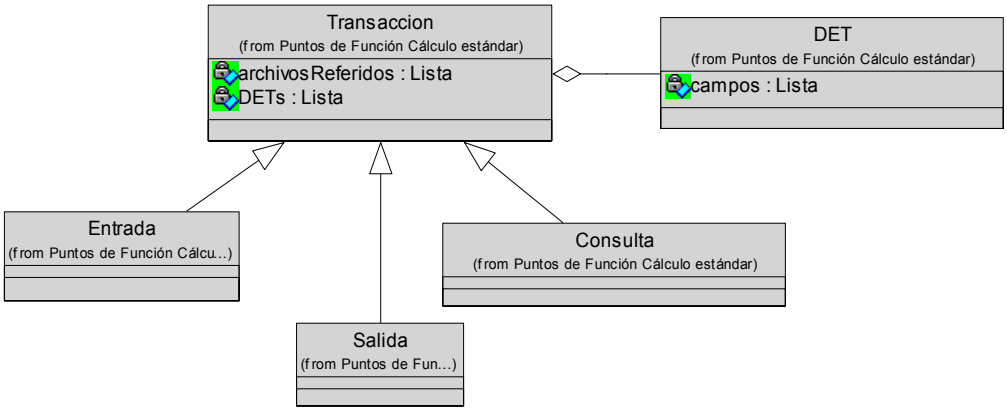

Figura 3.2.4. Diagrama UML para la clase *Transaccion*.

La clase *Transaccion* se deriva en las clases *Entrada*, *Salida*, y *Consulta*. Cada una de estas en representación de su tipo de funcionalidad correspondiente. Esta clase también contiene una lista de archivos referidos (*archivosReferidos*), y una lista de *DET*. Véase la sección…

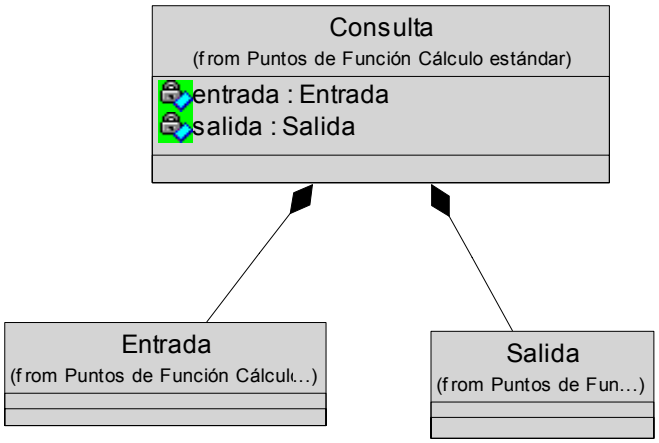

Figura 3.2.5. Diagrama UML para la clase *Consulta*.

La clase *Consulta* se compone de una parte de entrada (*atributo entrada*), y una parte de salida (*atributo de salida*). Véase la sección…

3.3. Cálculo de los puntos de función Mark II

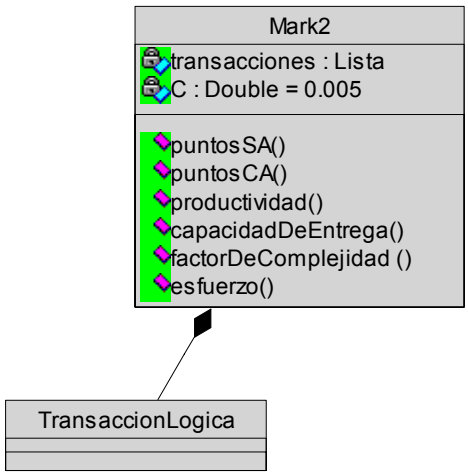

Figura 3.3. Diagrama de clases UML para el calculo de Mark II.

La clase *Mark2* contiene una lista de tracciones lógicas (*transacciones*), y la constante *C* del factor de complejidad, entre otros métodos que se describen por su nombre.

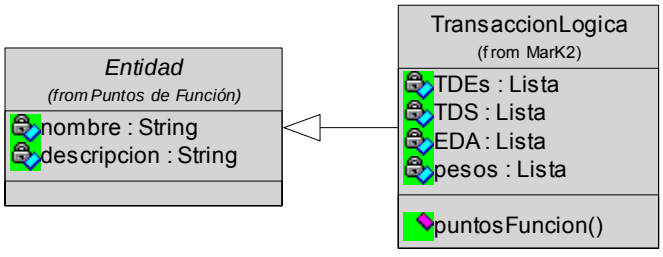

Figura 3.3.1. Diagrama UML para la clase *TransaccionLogica.*

La clase *TransaccionLogica* contiene la lista de pesos de complejidad de cada atributo que almacena, tipos de datos de entrada (*TDEs*), tipos de datos de salida (*TDS*) y entidades de datos afectadas (*EDA*).

Cada transacción lógica hereda de la clase *Entidad*, y calcula su contribución a la cuenta de puntos, puntosFuncion().

3.4. Utilización de análisis estadístico de datos históricos de métricas

3.4.1. Aplicación de la regresión lineal

La idea de utilizar la regresión lineal viene dada por las recomendaciones encontradas en varios trabajos investigativos []. Estos trabajos describen la regresión lineal como altamente aplicable a los datos recopilados durante el desarrollo de software.

Los datos de tamaño recolectados en proyectos pasados pueden ser utilizados para determinar los coeficientes de regresión lineal. Véase la sección 2.5 del Capítulo 2.

La intención es obtener estimaciones de LOC que sirvan como entrada para los modelos empíricos de estimación, por ejemplo COCOMO.

Un recurso importante para las estimaciones del proyecto es analizar la variación de los datos [] que se recolectan y que se utilizan para derivar estimaciones. De esta manera se podría determinar si los datos históricos que utilizamos predicen de alguna manera los aspectos a estimar.

3.5. Nivel de organización de los proyectos de software

Se destaca la importancia del nivel de organización y disciplina de los proyectos de software en cuanto a su comportamiento a la hora de desarrollar un determinado tipo de producto. Si los proyectos se desarrollan de forma caótica, no se recolectan medidas del producto, el personal cambia frecuentemente, ni son gestionados; no habrá oportunidad para la predecibilidad de los aspectos a estimar.

Lo anterior justifica el hecho de que para realizar buenas estimaciones es necesario contar con buen nivel de organización del proyecto. Un nivel alto en la escala CMM para una organización desarrolladora de software garantiza una alta predecibilidad de los aspectos a estimar por cuanto su comportamiento en el desarrollo de software es uniforme y disciplinado.

Es necesario crear una infraestructura para la gestión, planificación y estimación. Cuando se habla de infraestructura no solamente se refieren los aspectos anteriores, sino también el desarrollo de herramientas de software para cada uno de los métodos a utilizar.

CONCLUSIONES

Como conclusión de este trabajo se plantea la aplicación de los métodos de estimación del tamaño del software expuestos, y la adopción de las ideas dadas sobre la vista estructural del software a estimar.

Cada uno de los métodos de estimación descritos requiere ciertos datos de entrada que se hacen disponibles en distintos momentos del desarrollo del software (fases), teniéndose en cuenta la disponibilidad de estos para su aplicación.

Al inicio del proyecto los métodos de estimación deben partir de las especificaciones del producto a construir y la delimitación del ámbito del software. En estos primeros momentos no existe mucho detalle sobre como se desarrollará el producto. La única forma de hacer estimaciones cuantitativas es por analogía con datos históricos de proyectos anteriores. Téngase en cuenta:

- Puntos de Función
- Descomposición funcional del problema
- Estimación para proyectos orientados a objetos

Después de un diseño preliminar del software ya se han identificado algunos componentes del producto y los subsistemas ha desarrollar, los métodos de estimación pueden partir de estos datos para ofrecer resultados más precisos. Puede aplicarse:

- Adopción de un modelo de seguimiento del tamaño del software para desarrollo incremental
- Método de Wideband – Delphi
- Método difuso de Putnam
- Método de componentes estándar

Contando con un diseño detallado del software se tiene las especificaciones de los objetos a desarrollar e información asociada, se puede aplicar fácilmente la estimación utilizando el método PROBE.

No existe particular dificultad para la automatización de los métodos de estimación expuestos, la dificultad radica en como se organiza el proyecto de software para realizar las estimaciones.

Es importante crear un conjunto de condiciones necesarias previas a la aplicación de cualquier método de estimación, establecer una línea base de datos de métricas, y garantizar una buena gestión de los proyectos de software.

RECOMENDACIONES

1. Se recomienda realizar la implementación de las siguientes propuestas:

- Crear un generador de rangos de tamaño para el método difuso de Putnam.
- **Herramientas estadísticas.** Ejemplo: cálculo del promedio, media, varianza, deviación estándar, distribuciones probabilísticas, valor esperado, etc.
- Crear una librería para el manejo de componentes estándar.
- Dar soporte automatizado al método PROBE y otros métodos propuestos.

2. Realizar un entrenamiento y capacitación de las personas encargadas de realizar las estimaciones.

3. Continuar el estudio y perfeccionamiento de los métodos de estimación de tamaños de productos de software.

4. Realizar un estudio profundo de la tipología del software a fin de caracterizar a esta entidad en términos de las características generales que se identifiquen, y de las relativas a un tipo específico.

5. Por último se recomienda dar a conocer los resultados de este trabajo, teniéndolo como referencia para la futura investigación del tema, y guía para los proyectos productivos de la UCI.

GLOSARIO

Ámbito del producto (software): identifica los datos primarios, funciones y comportamientos que caracterizan al producto, e intenta abordar estas características de una manera cuantitativa (PRESSMAN 2002).

Aplicación (informática): programa informático diseñado para facilitar al usuario la realización de un determinado tipo de trabajo… Suele resultar una solución informática para la automatización de ciertas tareas complicadas como puede ser la contabilidad o la gestión de un almacén (ENCARTA 2007).

B.A: la **Licenciatura en Artes (B.A)** es el título tradicional de artes liberales que pone al estudiante en contacto con una amplia variedad de disciplinas—literatura, historia, ciencias sociales y ciencias de laboratorio—antes de la exigencia de especialización.

Las "Artes" aluden al estudio de un amplio espectro de disciplinas y no a las materias de estudio (*Aprendiendo sobre títulos universitarios* 2007).

Base de datos: cualquier conjunto de datos organizados para su almacenamiento en la memoria de un ordenador o computadora, diseñado para facilitar su mantenimiento y acceso de una forma estándar (ENCARTA 2007).

Boehm, Barry: graduado de **B.A** de la Universidad de Harvard en 1957. Durante su amplio desempeño profesional se le han conferido gran variedad de distinciones. Sus contribuciones al campo (Ingeniería del Software) incluyen el modelo constructivo del coste (COCOMO), el modelo espiral del proceso del software, el acercamiento de la teoría W (ganar-ganar) a la determinación de la gerencia y de los requisitos del software y a dos ambientes avanzados de la tecnología de dotación lógica: el sistema y el Quantum de la productividad del software de TRW (WIKIPEDIA 2007a).

Se recomienda consultar (GROUP 2007).

Dato: representación simbólica (numérica, alfabética, etc.), de un atributo o característica de una entidad. El dato no tiene valor semántico (sentido) en sí mismo, pero convenientemente tratado (procesado) se puede utilizar en la realización de cálculos o toma de decisiones (WIKIPEDIA 2007b).

Domino del sistema: contexto y características actuales del sistema sobre el cual se va a efectuar el desarrollo del nuevo proyecto de software (SANDRA PATRICIA FORIGUA PULIDO 2006).

Estándar: en informática, conjunto de especificaciones técnicas utilizadas para unificar el desarrollo de *hardware* o de software (ENCARTA 2007).

Estimación: Aprecio y valor que se da y en que se tasa y considera algo (ENCARTA 2007).

Ingeniería de Software: es la rama de la ingeniería que crea y mantiene las aplicaciones de software aplicando tecnologías y prácticas de las ciencias computacionales, incluye el manejo de proyectos y otros campos (WIKIPEDIA 2007c).

Interfaz: punto en el que se establece una conexión entre dos elementos, que les permite trabajar juntos. La interfaz es el medio que permite la interacción entre esos elementos (ENCARTA 2007).

Medición: es el acto de determinar una medida (E-CLASES).

Medida: proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto. Pueden ser directas, por ejemplo, número de líneas de código, número de errores encontrados, etc., o pueden ser indirectas, por ejemplo, funcionalidad, calidad, complejidad, etc. (E-CLASES)

Método: procedimiento que se aplica para llegar a un fin determinado (SANDRA PATRICIA FORIGUA PULIDO 2006).

Métrica[21]: es una medida cuantitativa del grado en que un sistema o proceso posee un atributo dado. Por lo general relaciona una o más medidas, p.e. número de errores encontrados por cada mil líneas de código (E-CLASES).

Paradigma (modelo) recursivo/paralelo: es un modelo para el desarrollo de software orientado a objetos. Se describe de la siguiente manera (BERARD 1993):

- Descomponer sistemáticamente el problema en componentes altamente independientes.
- Aplicar de nuevo el proceso de descomposición a cada uno de los componentes independientes para, a su vez, descomponerlos (la parte recursiva).
- Conducir este proceso de reaplicar la descomposición de forma concurrente sobre todos los componentes (la parte paralela).
- Continuar este proceso hasta cumplir los criterios de finalización.

Adaptado de (PRESSMAN 2002).

Proceso (de desarrollo) de software: es el conjunto de técnicas y procedimientos que nos permiten conocer los elementos necesarios para definir un proyecto de software (*Proceso de Desarrollo de Software* 2007).

Proceso recursivo/paralelo: proceso que aplica los principios del **modelo recursivo/paralelo.**

Cada iteración de un proceso recursivo/paralelo requiere planificación, ingeniería (análisis, diseño, extracción de clases, prototipado y pruebas) y actividades de evaluación (PRESSMAN 2002).

[21] Existe cierta polémica en la definición de este término, véase por ejemplo (MATEUS FERREIRA 2006).

Producto: cualquier software que será construido a petición de otros (PRESSMAN 2002).

Programa (informático): conjunto de instrucciones escritas en un lenguaje de programación para su ejecución en un ordenador o computadora. Por lo general, el término implica una entidad autocontenida, a diferencia de una rutina o una biblioteca (ENCARTA 2007).

Proyecto: un proyecto es una acción en la que recursos humanos, financieros, y materiales se organizan de una forma para acometer un trabajo único. En este trabajo dadas unas especificaciones y dentro de unos limites de costes y tiempo, se intenta conseguir un cambio beneficioso dirigido por unos objetivos cualitativos y cuantitativos (MORENO).

Proyecto de Desarrollo de Software: es un proyecto concebido para el desarrollo de un software. También puede denominarse **Proyecto de Software.**

Putnam, Lawrens (Larry): renombrada autoridad en las disciplinas de medición y estimación del software. Se recomienda consultar (*Lawrens (Larry) Putnam* 2007).

Rand Corporation (Corporación RAND): es un **think tank** norteamericano formado, en un primer momento, para ofrecer investigación y análisis a las fuerzas armadas norteamericanas.

Desde entonces la organización de esta corporación ha cambiado y actualmente también trabaja en la organización comercial y gubernamental de los Estados Unidos [...] Algunos consideran que el nombre de la corporación es un acrónimo de la frase "Reserch and Development" ("investigación y desarrollo") (WIKIPEDIA 2007e).

Requerimiento: características que el **producto** debe poseer (SANDRA PATRICIA FORIGUA PULIDO 2006).

Requerimientos funcionales: son aquellos que describen interacciones entre el sistema y su ambiente (usuario, otros sistemas) independientemente de su implementación (SANDRA PATRICIA FORIGUA PULIDO 2006)**.**

Sistema (informático): es la síntesis de hardware, software y de un soporte humano (WIKIPEDIA 2007f).

Software: todos los componentes intangibles de un ordenador o computadora, es decir, al conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica, en contraposición a los componentes físicos del sistema (hardware) (WIKIPEDIA 2007g).

Tamaño del software: el tamaño de un producto se software es un indicador de la amplitud y profundidad del conjunto de prestaciones que incorpora, así como de la dificultad y profundidad del programa (*Guía de Aprendizaje - Tema 8.Gestión de Costes en Proyectos Software.*).

Técnica: procedimientos y habilidad para hacer uso de estos en un arte u oficio (SANDRA PATRICIA FORIGUA PULIDO 2006).

Think tank: un think tank es una institución investigadora u otro tipo de organización que ofrece consejos e ideas sobre asuntos de política, comercio e intereses militares. El nombre proviene del inglés, por la abundancia de estas instituciones en Estados Unidos, y significa "depósito de ideas". Algunos medios en español utilizan la expresión "usina de ideas" para referirse a los think tank (WIKIPEDIA 2007h).

REFERENCIAS BIBLIOGRAFICAS

1. Disponible en: http://www.fing.edu.uy/inco/cursos/gestsoft/ensayo_parcial.doc
Cálculo de PF Disponible en: http://alarcos.inf-cr.uclm.es/doc/pgsi/doc/teo/8/pgsi-t8pf.xls

Capítulo 3. PSP 0 y PSP 0.1 Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/pelaez_r_jj/capitulo3.pdf

DURÁN, S. *Análisis de Puntos de Función SG. Software Guru*, 2005

E-CLASES. *Ingenieria de Software IV. Métricas de Software*. Disponible en: http://eclases.tripod.com/id14.html

ENCARTA, M., © 1993-2006 Microsoft Corporation. Reservados todos los derechos., 2007. [Disponible en:
Estimating of Software Size - Part I 2003. [Disponible en: http://www.engr.sjsu.edu/ahambaba/course/Estimating%20of%20Software%20Size%20-%20Part%20I%20-%20Fall%202003.ppt

Estimating of Software Size - Part II 2003. [Disponible en: http://www.engr.sjsu.edu/ahambaba/course/Estimating%20of%20Software%20Size%20-%20Part%20II%20-%20Fall%202003.ppt

Fundamentos de Ingenieria de Software. Módulo 2. 2007. [Disponible en: http://148.202.148.5/cursos/cc321/fundamentos/unidad2/tema2_2.html

GIRALDO, O. P. *Métricas, Estimación y Planificación en Proyectos de Software*

Documento de la Universidad de Guadalajara Disponible en: http://www.willydev.net/Descargas/WillyDEV_PlaneaSoftware.Pdf

GREENE, J. *Measures for Excellence. Sizing and controlling Incremental Software Development*, QSM Ltd. Disponible en: http://www.qsm.com/sizing.pdf

GROMPONE, J. *Gestión de Proyectos de Software* 1996. [Disponible en: http://www.itapebi.com.uy/pdfs/GPS.pdf

Guía de Aprendizaje - Tema 8.Gestión de Costes en Proyectos Software. . Disponible en: http://alarcos.inf-cr.uclm.es/doc/pgsi/doc/teo/8/pgsi-t8.pdf

HUMPHREY, W. *A Discipline for Software Engineering*. Addison-Wesley, 1995. p. *Reading.*

IFPUG. Disponible en: http://www.ifpug.org.

INFORMÁTICA, C. S. D. Métrica Versión 3. Fase 1: Interfaces. Gestión de Proyectos Software (Proceso GP) 1999: 35-38

Instructivo para la Cuenta de Puntos Función. Disponible en: http://www.pdt.gub.uy/files/6.2.5%20-%20puntosFuncion.pdf

LORENZ, M. Y. J. K. *Object-Oriented Software Metric*. Prentice-Hall, 1994. p. p. 356-357.

MARCO RONCHETTI, G. S., WITOLD PEDRYCZ, BARBARA RUSSO. *Early Estimation of Software Size in Object Oriented Enviroments. A Case Study in a CMM Level 3 Software Firm*, University of Trento, 2004. [Disponible en: http://eprints.biblio.unitn.it/archive/00000517/01/EstimationOfSoftwareSize.pdf

MATEUS FERREIRA, F. G., FRANCISCO RUIZ A, MANUEL F. BERTOA, CORAL CALERO, ANTONIO VALLECILLO, MARIO PIATTINI, BEATRIZ MORA. *Departamento de Tecnologías y Sistemas de Información. Medición del Software Ontología y Metamodelo.* , Universidad de Castilla-La Mancha (UCLM), España. , 2006. [Disponible en: http://titan.inf-cr.uclm.es:8081/tsi/informes/uclm.tsi.001.pdf

MCCONNELL. *Rapid Development: Taming Wild Software Schedules* 1996. [Disponible en: http://dspace.mit.edu/html/1721.1/36380/1-264JFall-2002/NR/rdonlyres/Civil-and-Environmental-Engineering/1-264JDatabase--Internet--and-Systems-Integration-TechnologiesFall2002/0DA5FE6A-8E0B-4D13-824D-F35390B8A62D/0/1264_lecture_3_F2002.pdf

Medición del Software Disponible en:
Métricas en el desarrollo del Software. Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/gonzalez_d_h/capitulo4.pdf

MORENO, A. M. *Estimación de Proyectos Software*. Disponible en: http://www.inf.uach.cl/rvega/asignaturas/info265/estim_sexto.pdf

NAWROCKI, J. *Personal Software Process. Software Size Estimation (I)* 1999. [Disponible en: www.cs.put.poznan.pl/~nawrocki/mse/psp
PHILLIPS, D. *The Software Project Manager´s Hand-book*. IEEE Computer Society Press, 1998. p.

PRESSMAN, R. S. *Ingeniería del Software: Un Enfoque Práctico* Quinta Edición McGraw-Hill, 2002. p.

Proceso de Desarrollo de Software 2007. [Disponible en: http://cursos-gratis.emagister.com.mx/proceso-desarrollo-software-cursos-317375.htm

RENDÓN, J. P. G. *Ingeniería del software (Una aproximación a la medición de la calidad)*, 2007. [Disponible en: http://www.monografias.com/trabajos15/ingenieria-software/ingenieria-software2.shtml#METRICAS

SANDRA PATRICIA FORIGUA PULIDO, O. A. B. G. *Propuesta de un modelo de análisis para estimación del tamaño del software y gestión de costos y riesgos a partir de requerimientos funcionales*, Pontificia Universidad Javeriana. Facultad de Ingeniería. Carrera de Ingeniería de Sistemas. , 2006. [Disponible en: http://www.pegasus.javeriana.edu.co/~riesgors/propuesta.pdf

Software Estimation
V., M. P. *Informe Final "Métricas Funcionales"*, Universidad de Talca 2006. [Disponible en: http://ftp.utalca.cl/asignaturas/isw/isw2/icc/staff/p-isw2-icc/metricas-funcionales.doc

WIKIPEDIA Dato, 2007a.
---. *Ingenieria de software* 2007b. [Disponible en: http://es.wikipedia.org/wiki/Ingenier%C3%ADa_del_software

---. *Métrica de punto función* 2007c. [Disponible en: http://es.wikipedia.org/wiki/M%C3%A9trica_de_punto_funci%C3%B3n

---. *Sistema Informático*, 2007d. [Disponible en: http://es.wikipedia.org/wiki/Sistema_inform%C3%A1tico

---. *Software*, 2007e. [Disponible en: http://es.wikipedia.org/wiki/Software