Métricas para evaluar la complejidad de los productos software

Lianny O'farrill Fernández y Ana María García Pérez

Edición: Liset Ravelo Romero
Corrección: Fernando Gutiérrez Ortega
Diagramación: Roberto Suárez Yera

Editorial Feijóo, 2012

ISBN: 978-959-250-822-4

EDITORIAL
Feijóo

Editorial Samuel Feijóo, Universidad Central "Marta Abreu" de Las Villas, Carretera a Camajuaní, km 5 ½, Santa Clara, Villa Clara, Cuba. CP 54830

Índice

Tabla de contenido

Índice 3

Introducción 5

1.1 Estimación. Objetivos e importancia. 7

1.2 Complejidad del software 9

1.3 Métodos para medir la complejidad del software 10

1.3.1 Métodos basados en el criterio de expertos 10

1.3.2 Las métricas como instrumento para abordar la complejidad 13

Estructuras de control del programa 13

Medidas híbridas 13

Métricas de tamaño 14

Intervalo entre referencia a datos 14

Par de uso segmento-global 14

Medida Q de Chapin 15

Número Ciclomático 16

Extensión de Myres al número ciclomático 16

Métrica de Hansen 16

Métrica de Oviedo 16

Líneas de Código Fuente (LOC) 17

Métricas de Halstead 19

Puntos de Función (PF) 20

Puntos de Objetos 22

Puntos de Casos de Uso (UCP) 23

UCP = $i = 12j = 13Wij * Iij$ *(I)*.....23

Transacciones, objetos de entidad y caminos.....24

Puntos de características.....26

Mark II.....26

3D–FUNCTION POINTS.....27

PUNTOS COMPLETOS DE FUNCIÓN (FFP).....28

COSMIC-FFP.....28

SLIM. Ecuación del Software.....30

COCOMO.....33

Conclusiones.....35

Referencias bibliográficas.....36

Introducción

La industria del software aunque es relativamente nueva ha tenido un impacto importante en la sociedad y en los últimos años ha marcado pautas en el desarrollo de las organizaciones empresariales.

Los proyectos de desarrollo de software se diferencian de otros proyectos de ingeniería tradicional en la naturaleza lógica del producto software. Este se desarrolla, no se fabrica en un sentido clásico. En todos los proyectos de ingeniería la buena calidad se adquiere mediante un buen diseño, pero en el caso del software, la etapa de construcción incide pobremente en su calidad, no sucede así en la construcción de los hardwares o de una obra civil. Otra diferencia radica en que no se estropea, el paso del tiempo o males del entorno no inciden en el aumento de la tasa de fallas. Por lo cual no se puede gestionar un proyecto de desarrollo de software como si se tratara de un proyecto de fabricación de un bien tangible (Varas, 2000).

El proyecto de software es un caso particular de proyecto donde existe una gran incertidumbre sobre: el resultado final, su costo, sus riesgos, el esfuerzo y el tiempo que implica su desarrollo. El producto final es intangible desde el punto de vista físico, su valor depende no solo de su corrección, sino del momento en que se pone en servicio, la calidad apreciada por el usuario, su facilidad de uso, mantenimiento y extensión.

Una de las áreas de la ingeniería de software es la Gestión de Software y dentro del proceso de gestión, una de las actividades cruciales es la planificación, la cual se basa en una buena estimación del esfuerzo requerido para realizar el proyecto, su duración cronológica y el costo.

Aunque se trabaja en la mejora del proceso de producción de software, aún existen retos para los especialistas encargados del desarrollo de ese tipo de sistemas. Uno de los retos es el proceso de estimación de variables como el tiempo, el esfuerzo y los costos.

Uno de los problemas de los modelos de estimación actuales es que son dependientes del juicio experto y requieren gerentes con experiencia para aplicarlos. Los gerentes con experiencia cada vez son más escasos y la tendencia actual es desplazarse hacia metodologías ágiles con equipos altamente reducidos, motivados y comprometidos con la realización del proyecto.

Independizar la estimación (no la interpretación de los resultados) del juicio experto aumenta la credibilidad de la industria, facilita la relación entre clientes y desarrolladores aumentando su

transparencia. Puede ser el comienzo de metodologías de gestión de contratos que no se basen en un precio cerrado en el momento en que menos se sabe del proyecto, sino en una metodología de cálculo de acuerdo con las condiciones de ejecución del proyecto y la complejidad del sistema que se pretende construir, contribuye a poner en evidencia, sobre bases objetivas, los riesgos que introducen las fechas fijadas con criterios no técnicos, y apoya la gestión conjunta de los cambios, la discusión sobre bases objetivas y claras para todos los actores, de plazos, alcance de los proyectos y los riesgos derivados de ellos (Salvetto, 2006).

Un factor que incide de manera directamente proporcional en los valores estimados de tiempo, costo y esfuerzo es la complejidad. Mientras más complejo es el producto por desarrollar, mayor será el tiempo que se invertirá en su construcción, el esfuerzo que demandará su realización y los costos asociados. A partir de determinar el nivel de complejidad que demanda construir un producto de software, se puede tener una idea del tiempo, el costo y el esfuerzo que se deben invertir en su desarrollo.

¿Cómo medir el nivel de complejidad del proceso de desarrollo de un producto de software?

Para medir el nivel de complejidad del desarrollo de un producto de software se pueden seguir dos modos fundamentales: a partir del criterio de expertos o especialistas y con el uso de métricas. A continuación se explica brevemente en qué consiste cada uno de ellos:

Criterio de expertos o de especialistas: haciendo uso de sus conocimientos o experiencia en trabajos similares, los expertos se lanzan a estimar el nivel de complejidad del proceso. Esto es algo arriesgado, y como se mencionaba anteriormente, los expertos son escasos y los conocimientos de quienes se dedican a la construcción de software no siempre son suficientes para estimar la complejidad, sumándole a esto que cada producto de software es único, independientemente de por quién sea desarrollado o la similitud con otros productos del mismo tipo.

Por medio de métricas: se pueden definir métricas de complejidad teniendo en cuenta que todas las métricas de software definen de una u otra forma la medición de la complejidad; tales como el volumen, el tamaño, las anidaciones, el costo (estimado), la agregación, la configuración, y el flujo. Dentro de las métricas antes mencionadas, las más utilizadas para medir el nivel de complejidad son las métricas de tamaño (Nieto, 2008).

Las métricas de tamaño se pueden clasificar en dos grupos, las métricas que determinan el tamaño a partir del código fuente del sistema y las métricas de tamaño funcional. Desde ambas métricas se plantea una relación directamente proporcional con la complejidad, estableciendo que el desarrollo de cierto producto será tan complejo según sea su tamaño.

1.1 Estimación. Objetivos e importancia

El desarrollo de software es una tarea compleja. Las organizaciones necesitan habitualmente herramientas de software de tamaño medio o grande, y la complejidad para llevar a cabo con éxito dichos desarrollos crece exponencialmente con el tamaño del producto. Se hace, por tanto, necesaria una disciplina para poder realizar dichos desarrollos con éxito. Dicha disciplina es la Gestión de proyectos, encargada de organizar y administrar recursos de manera tal, que se pueda culminar todo el trabajo requerido en el proyecto dentro del alcance, el tiempo, y el costo definido.

La gestión de proyectos es el proceso por el cual se planifica, dirige y controla el desarrollo de un sistema aceptable con un costo mínimo y dentro de un período de tiempo específico (Varas, 2000).

El proceso de gestión del proyecto de software comienza con un conjunto de actividades que, globalmente, se denominan planificación del proyecto (Pressman, 1998).

Dentro de las actividades de la gestión están, la *planificación* encargada de predeterminar un curso de acción para alcanzar los objetivos organizacionales; la *organización* asume el arreglo de las relaciones entre las unidades de trabajo para el cumplimiento de los objetivos y el otorgamiento de responsabilidades; *la dotación del personal* (*staffing*) se responsabiliza de la selección y entrenamiento de personas para puestos en la organización, la *dirección* se encarga de la creación de una atmósfera que apoye y motive a las personas para alcanzar los resultados finales deseados y por último el *control* el establecimiento, medición y evaluación del desempeño de las actividades a través de los objetivos planeados (Varas, 2000).

Una de las actividades cruciales del proceso de gestión es la planificación. El objetivo de la planificación es proporcionar un marco de trabajo que permita al gestor hacer estimaciones razonables de recursos, costos y planificación temporal. Estas estimaciones se hacen dentro de un marco de tiempo limitado al comienzo de un proyecto de software, y deberían actualizarse regularmente a medida que progresa el proyecto.

Para todo proyecto de desarrollo de software, siempre que se estima se echa un vistazo al futuro y se acepta un grado de incertidumbre. Es por esto que dentro del contexto de las primeras etapas de un proyecto de software se encuentra que uno de los aspectos más críticos es la estimación. Este aspecto afecta a todo el proyecto y en especial las etapas de análisis y diseño, las cuales normalmente son inmediatas, en la mayoría de metodologías de desarrollo

de software, la fase inicial en la que se formula el proyecto y donde tradicionalmente se realiza la estimación.

El objetivo de la estimación es predecir las variables involucradas en el proyecto con cierto grado de certeza, trata de aportar una predicción de algún factor importante para la gestión de proyectos de software: tiempo, esfuerzo, cantidad de defectos esperados, entre otros, sin dejar de tener en cuenta que la incertidumbre y los riesgos son elementos inherentes a ella.

La estimación es llevada a cabo por el gestor del proyecto. La estimación y planificación temporal de un proyecto software requiere de experiencia, buena información histórica, o una buena técnica de estimación en la que se confíe plenamente por los resultados que arroje.

Aunque la estimación es más un arte que una ciencia, es una actividad importante que no debe llevarse a cabo de forma descuidada. Existen técnicas útiles para la estimación de costes y de tiempos. Y dado que la estimación es la base de todas las demás actividades de planificación del proyecto y sirve como una guía para una buena ingeniería del software, no es en absoluto aconsejable embarcarse sin ella (Pressman, 1998).

La estimación es importante no solo para predecir el valor de variables concretas dentro de un proyecto sino para determinar su viabilidad. No tiene sentido iniciar un proyecto que está destinado al fracaso por no contar con el tiempo, el esfuerzo o los recursos necesarios para llevarlo a cabo. En la actualidad son muchos los proyectos que fracasan, e incumplen sus plazos de entrega. La realidad de la industria de software, refleja que un alto porcentaje de las estimaciones realizadas para proyectos de software están lejanas de los resultados, del esfuerzo neto invertido en éstos. Según el *The Standish Group* en su publicación "Chaos Report 2009," el 24 % de los proyectos informáticos fueron cancelados, solo el 32 % terminaron en tiempo y dentro del presupuesto y el 44 % de los proyectos de software fueron concluidos después de la fecha estimada.

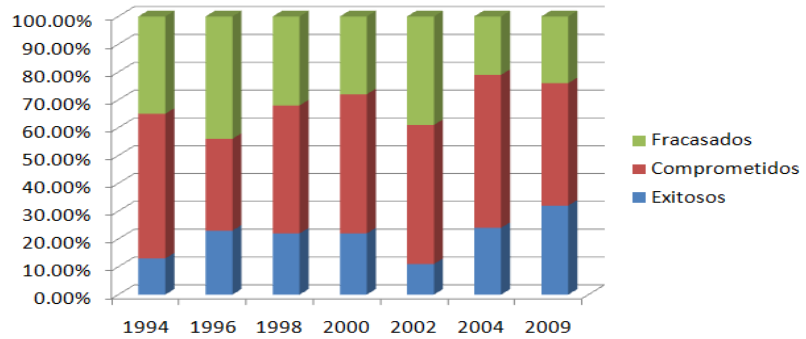

Fig1. Estado de los proyectos según (The Standish Group, 2009)

Los clientes y usuarios de un producto de software se sienten cómodos cuando ellos están seguros de que podrán usar la aplicación propuesta en un tiempo razonable; en este sentido, ellos se convierten en promotores del proyecto. Según (Varas, 2000) existen dos aspectos importantes para hacer que los usuarios se sientan cómodos: la planificación y los costos de los proyectos. La precisión de las estimaciones son importantes para:

- Realizar análisis de costo-beneficios y financieros.
- Realizar análisis de inversión de hardware y software.
- Proveer la base para la evaluación gerencial de múltiples proyectos.
- Servir de fundamento para los cronogramas, asignación de personal, gerencia de proyectos y definición de estructura.
- Evitar problemas como la renegociación de contratos, sobre tiempos, incrementos de los costos de los usuarios o de los costos de los proyectos.

Es importante reconocer la fuerte relación entre costo, cronograma y calidad. Estos tres aspectos están íntimamente relacionados y confrontados entre sí. De esta manera, se hace difícil incrementar la calidad sin aumentar el costo y/o el cronograma del software a desarrollar. Similarmente, el cronograma de desarrollo no puede reducirse dramáticamente sin deteriorar la calidad del producto de software y/o incrementar el costo de desarrollo. Los modelos de estimación juegan un papel importante ya que permiten equilibrar estos tres factores.

Un factor que incide de manera directamente proporcional en la estimación de los valores de tiempo, costo y esfuerzo es la complejidad, se plantea que mientras más complejo es el producto a desarrollar mayor será el tiempo que se invierta en su construcción, el esfuerzo que demanda su realización y los costos asociados. A partir de determinar el nivel de complejidad que demanda construir un producto de software se puede tener una idea del tiempo, el costo y el esfuerzo que se deben invertir en su desarrollo.

1.2 Complejidad del software

Por mucho tiempo el software se ha mostrado ante los ojos de los usuarios como algo simple de utilizar, para muchos de ellos existe la idea de que mediante la computadora todo es fácil, solo es cuestión de apretar un botón y ya está. Desafortunadamente, para un ingeniero de software, un pequeño capricho del usuario puede implicar varias horas de programación (Sommerville, 2006).

Esta complejidad del software también permanece oculta porque por naturaleza el software es abstracto, de modo que en ocasiones puede verse más como un servicio que como un producto. Hay autores que consideran que el software va más allá de un conjunto de

programas, y que debe incluir elementos como documentación de uso y de estructura del sistema (Booch, 2007).

Algunos aspectos que inciden en la complejidad del software son: el complicado contexto en el cual surge la necesidad o idea de construir el sistema, esta necesidad puede que no sea entendida por el analista como el cliente quisiera y esto llevar a un levantamiento de requerimientos erróneo. Otro aspecto es el hecho de que los clientes se sientan con el derecho de proponer nuevos requisitos, esto podría desencadenar cambios en la estructura del programa y aumentaría la complejidad de desarrollar el producto. La gestión del equipo encargado de desarrollar el sistema influye en gran medida en la complejidad, el gestor del proyecto debe ser una persona capacitada puesto que tiene la encomienda de dirigir a un grupo de personas que deben acoger ciertas herramientas, estándares y técnicas para lograr un proceso de desarrollo exitoso.

¿Cómo se mide la complejidad para desarrollar un producto de software?

Para medir el nivel de complejidad del desarrollo de un producto de software se pueden identificar dos modos fundamentales: a partir del criterio de expertos o especialistas y con el uso de métricas.

1.3 Métodos para medir la complejidad del software

1.3.1 Métodos basados en el criterio de expertos

Una forma de medir el nivel de complejidad del desarrollo de un producto de software es a partir del criterio de expertos o de especialistas, que haciendo uso de sus conocimientos y experiencia en trabajos similares se lanzan a estimar el nivel de complejidad del proceso de desarrollo del producto por construir, esto es algo arriesgado en tanto a que los expertos, como se mencionaba anteriormente en la introducción, son escasos y los conocimientos de quienes se dedican a la construcción de software no siempre son suficientes para estimar la complejidad, sumándole que cada producto de software es único independientemente de por quién sea desarrollado o la similitud con otros productos del mismo tipo de acuerdo con funcionalidades, entornos de desarrollos, lenguajes y otras cuestiones.

A continuación se muestran algunas técnicas que pueden ser útiles para medir la complejidad en este sentido.

1.3.1.1 Estimación por Juicio de Experto

Es una técnica de estimación por la cual el gerente del proyecto y los ingenieros de software usan sus experiencias para guiar las estimaciones de tiempo y de costos.

Cada tarea es definida en términos de los tipos de programas, más que como el resultado de la tarea. Los gerentes de proyecto y los ingenieros de software le asignan tiempo a cada programa, agregando los tiempos de análisis y diseño. Los costos son asignados de manera similar, cada tipo de programa es usado para definir el nivel de habilidades del programador deseado. Las ventajas de esta técnica son, que se basa en la experiencia para las estimaciones, ajusta las estimaciones al personal asignado y las estimaciones son hechas rápidamente y eficientemente, es capaz de tener en cuenta las diferencias entre las experiencias de proyectos anteriores y las nuevas técnicas, arquitectura o aplicaciones involucradas en el nuevo proyecto, el experto puede considerar también las condiciones excepcionales que envuelven el proyecto y que no pueden ser considerados en otros métodos como por ejemplo los métodos algorítmicos (Mendoza, 2009).

Según (Princich, 2009) la estimación de esfuerzos basada en juicio experto, es la técnica más aplicada y menos compleja de utilizar, a veces solo consiste en replantear preguntas.

Es una técnica confiable aunque bastante imprecisa. Generalmente los "expertos en la materia" incurren en la subestimación de esfuerzos, lo que hace a estimaciones generalmente riesgosas.

Según Steve MacConnell (MacConnell, 2006) menciona los trabajos de investigación de Michiel van Genuchten, que afirma que las estimaciones realizadas por los desarrolladores tienen un factor de optimismo de entre el 20 % y el 30 %, esto quiere decir que los desarrolladores de software estiman que pueden hacer las cosas un 30 % más rápido de lo que realmente pueden.

1.3.1.2 La técnica Delphi

La técnica Delphi se basa en la obtención de un consenso de un grupo de expertos, que expresan sus opiniones y ofrecen estimaciones sobre el proyecto en cuestión (Alvarez, 2007).

Los expertos expresan sus opiniones mediante unos formularios que le son entregados y que rellenan de manera totalmente anónima. Los cuestionarios contienen cada una de las estimaciones realizadas a nivel personal por cada componente del grupo.

Estos cuestionarios son entregados al coordinador del proceso, quien se encarga de comunicarle a cada experto la opinión de los demás, junto con datos estadísticos sobre todas las estimaciones entregadas. Una vez que cada estimador posee esta información, vuelven a rellenar los cuestionarios y los entregan nuevamente al coordinador. Este proceso se repetirá hasta que el coordinador encuentre un consenso y una opinión generalizada acerca de la estimación del proyecto.

Sin embargo, ofrece una serie de desventajas, relacionadas con la elaboración de los cuestionarios y la selección de los expertos.

1.3.1.3 Analogía

Según (Alvarez, 2007) la estimación basada en analogía es el proceso por el cual se localizan uno o más proyectos similares al que está siendo desarrollado y se pueden realizar estimaciones a partir de ellos.

Es fundamental en este proceso medir el grado de similitud del proyecto bajo examen con respecto a los almacenados en la base de datos de históricos. Así se pueden identificar los proyectos más parecidos al actual y estimar a partir de ellos.

No obstante, resulta obvio pensar que es prácticamente imposible que exista uno o varios proyectos exactamente iguales al que se desea estimar, por lo que no se debe utilizar directamente el esfuerzo asociado con estos proyectos. La solución aportada consiste en aplicarle algún tipo de ajuste, en función de la distancia existente entre dicho proyecto obtenido como el más parecido y el proyecto sobre el que se va a realizar una estimación.

Para ajustar estos valores de esfuerzo de proyectos ya completados existen multitud de técnicas, siendo una de las más apropiadas la búsqueda del conjunto óptimo de factores por tener en cuenta en la estimación final. Los algoritmos más potentes en este campo son los genéticos.

En el proceso para la estimación por Analogía, lo primero es determinar cuáles de los proyectos que están almacenados es el más parecido con el que se está analizando, para eso hay un conjunto de técnicas, algunas se expondrán a continuación.

La distancia Euclídea $d(Pxi, Pyi) = \sqrt{\sum_{i=1}^{m} (Pxi - Pyi)^2}$: mediante esta fórmula de distancia se puede conocer el conjunto de proyectos más parecido con el que está bajo examen.

Ajuste mediante algoritmos genéticos. Este enfoque pretende escoger el mejor conjunto de características de los proyectos más cercanos con que se quiere examinar, y realizar las estimaciones con ellas. Para seleccionar dicho grupo se deben calcular los errores producidos, estimar con sus componentes, y buscar aquel conjunto que minimice este error en la estimación.

Ajuste por "regresión hacia la media": la "regresión hacia la media" es un fenómeno estadístico ligado con aquellas variables cuyo coeficiente de correlación es inferior al 100 %. Antes de realizar un ajuste por regresión hacia la media (RTM), deben valorarse dos

importantes aspectos: la exactitud de la estimación hecha a partir de los proyectos análogos, y las situaciones extremas que pueden presentar sus características.

Ventajas de este método según (Alvarez, 2007):

- bastante preciso si se dispone de datos de proyectos previos,
- es un método de razonamiento similar con la mente humana,
- es apropiado para las situaciones en las que el dominio es difícil de modelar,
- no se necesita tener mucha información, ni avanzada del nuevo proyecto ya que el peso recae en los proyectos ya completados,

Inconvenientes del método según (Alvarez, 2007):

- imposible de realizar si no se han abordado proyectos comparables,
- necesita el mantenimiento sistemático de una base de datos.
- el proyecto establecido puede no ser apropiado para la estimación,
- se debe ajustar el valor del esfuerzo del nuevo proyecto,
- se puede caer en la subjetividad.

1.3.2 Las métricas como instrumento para abordar la complejidad

Según Mills y Dyson: "No se puede controlar lo que no se puede medir" (Mills, 1990). Las métricas de complejidad de software ofrecen una base objetiva para identificar estructuras y técnicas que permiten producir programas de menor o mayor complejidad. Son medidas cuantitativas de ciertas características de un proyecto de desarrollo. Pueden medir objetos como:

- productos (como el código o la documentación),
- el proceso de desarrollo como tal (aspectos de las actividades del desarrollo),
- el dominio del problema (como las telecomunicaciones, los sistemas de tratamiento de información y el control de procesos),
- las características ambientales (las personas, las organizaciones y las herramientas)

Los tipos de métricas que inciden en la complejidad del software se enfocan con:

estructura y flujo de Control:

- Intervalo entre referencia a datos,
- Par de uso segmento-global,
- Medida Q de Chapin,

estructuras de control del programa:

- Número-ciclomático,
- Extensión de Myres al número-ciclomático

medidas híbridas:

- métrica de Hansen,
- métrica de Oviedo,

métricas de tamaño:

- número de líneas de código,
- métricas de Halstead,
- Metricas de tamañofuncional.

1.3.2.1 Métricas de estructura y flujo de control

Un factor importante en la complejidad del software viene dado por la forma como se manejan los datos dentro del programa. Con igualdad de otros factores será más complejo aquél en que los datos se manejen de forma más complicada. Según los criterios que se tienen en cuenta para medir esta complicación se definieron diferentes métricas.

Intervalo entre referencia a datos

Según (Sáez, 2009) un intervalo de referencia se define como: el número de sentencias que hay en el listado de un programa entre dos referencias inmediatas al mismo identificador de variable. El cálculo de la métrica consiste en contar la cantidad de intervalos entre referencias que sean mayores que cierto intervalo.

Esta técnica se apoya en la idea de que cuanto más dispersas por el código estén las referencias de una variable, más difícil de entender será el comportamiento de esa variable. Con una gran dispersión, el programador tendrá que tener en la cabeza los posibles cambios de valor en zonas del listado muy separadas entre sí. Sin embargo, con dispersión pequeña podrá centrarse más en el código adyacente que está estudiando. Por tanto, cuando las referencias de las variables se separan el programa será más difícil de mantener y los efectos colaterales indeseados, difíciles de evitar. En general, el software tendrá una mayor complejidad.

Par de uso segmento-global

En (Sáez, 2009) se explica que esta métrica consiste en que dado un segmento de código p y una variable global r, se define el par de uso segmento-global (p, r) como un indicador de que el segmento p usa la variable r. Dicho de otra forma r es accedida dentro de p. Se define el par de uso real (AUP) como el número de veces que un módulo utiliza una variable global. Y el par de uso potencial (PUP) como el número de veces que un módulo podría acceder una variable global (se entiende que un módulo p podría acceder a una variable r si p se encuentra en el ámbito de r). Por último, se define el porcentaje relativo de uso real (RUP) como AUP/PUP. Gracias a esta fórmula se obtiene una medida aproximada de cuánto se usan datos globales dentro de un segmento arbitrario de código.

Esta medida intenta dar una idea de la cantidad de veces que un segmento arbitrario de programa accede a una variable global. Se supone que si se hacen muchos accesos de este tipo, es fácil que el programador cometa errores, y se produzcan efectos colaterales indeseados en un segmento cuando cambia el valor de una variable global en otro. Estos errores son especialmente posibles en la fase de mantenimiento y modificación del programa.

Medida Q de Chapin

En esta métrica las medidas son tratadas según su uso dentro de cada segmento de código. Para ello son divididos en cuatro categorías: (Chapin, 1979):

- datos de tipo P: son los datos de entrada necesarios para que el segmento de código considerado produzca una salida.
- datos de tipo M: datos que se crean o cuyo valor cambian dentro del segmento,
- datos de tipo C: datos que se usan para ejercer un papel de control dentro del segmento,
- datos de tipo T: los que pasan dentro del segmento sin experimentar cambios.

Un mismo dato puede representar diferentes papeles dentro del mismo segmento de código, en este caso se contará una vez en cada clase que pueda pertenecer.

Chapin considera que no todos los tipos de datos descritos contribuyen con igual cantidad a la complejidad global del código que se esté estudiando. Los datos de tipo C serán los que más complejidad producen, ya que deciden cuál será el curso de la ejecución y qué módulos serán llamados. Luego vendrían los de tipo M y los de tipo P (que suelen ser usados para modificar los M). Por último los datos de tipo T casi no contribuyen con la complejidad ya que simplemente pasan por el módulo. A partir de estas consideraciones, se asigna un factor de ponderación diferente con cada uno de los tipos: 3 para el tipo C, 2 para el M, 1 para el P y 0,5 para el tipo T.

El conjunto de Pasos a seguir para el cálculo de la métrica se puede ver en (Chapin, 1979).

1.3.2.2 Métricas relacionadas con la estructuras de control del programa

Las posibilidades de que el flujo de ejecución de un programa siga diversos caminos según se cumplan o no ciertas condiciones, aumenta de una forma decisiva la dificultad para entender lo que hace el programa en cada una de las situaciones que se pueden dar.

Normalmente la complejidad debida al flujo de control se mide contando las transferencias de control que pueden darse en el código (teniendo en cuenta también la longitud total del programa) o estudiando sus interrelaciones. Las métricas que se presentan a continuación fueron extraídas de (Sáez, 2009).

Número Ciclomático

McCabe propone una medida de complejidad para un programa, basada en su grafo de control. Esta métrica ha sido ampliamente aceptada, por la facilidad para calcularse y para asimilar los resultados que ofrece.

Para calcular la complejidad ciclomática se utiliza la siguiente expresión matemática

V(G) = a – n+ 2c Donde:

a: número de arcos

n: número de nodos

c: componentes conectados

El número ciclomático puede entenderse como el número mínimo de caminos necesarios para, mediante combinaciones, construir cualquier otro camino presente en el grafo (Sáez, 2009).

Extensión de Myres al número ciclomático

La métrica de Myres surge a partir de que las sentencias con condiciones compuestas son más complejas que las que tienen una sola condición y sin embargo ambas tiene el mismo grafo y por tanto la misma complejidad ciclomática según McCabe. Myres decide tener en cuenta esta diferencia, midiendo la complejidad como un intervalo y no como un número. Para ello toma como límite inferior del intervalo el número de sentencias de decisión más uno y como superior, el número de condiciones individuales también más uno.

1.3.2.3 Métricas híbridas

La mayoría de las métricas tienden a medir sólo cierta parte de los aspectos que contribuyen a que un programa sea complejo. Pero muchas veces es interesante caracterizar de una forma más global la complejidad de un programa, para ello han de considerarse a la vez varias propiedades del código.

Métrica de Hansen

Hansen propone en (1978) una combinación del número ciclomático de McCabe con una medida del número de operandos. La métrica de Hansen es un par ordenado (m_1, m_2) donde:

m_1: es el número de sentencias alternativas (IF, CASE, etc.) o iterativas (DO, WHILE)

m_2: es el número de operadores del programa, definidos de una manera similar al $n_1$de Halstead.

De esta manera tan sencilla se consigue caracterizar los fragmentos de código con dos números que dan una idea del *flujo* de control (m_1) y de la cantidad de información total que contiene (m_2). Además, al estar basada en dos medidas ampliamente estudiadas y probadas puede asegurarse la validez de la métrica.

Métrica de Oviedo

Esta métrica intenta medir simultáneamente la complejidad de acuerdo con el flujo de datos y al flujo de control. Para ello se propone la siguiente fórmula: C = a*cf* + b*df* donde:

cf: representa la complejidad del flujo de control.

df: representa la complejidad del flujo de datos.

a, b son factores de peso para dar importancia a uno u otro de los aspectos medidos, en una primera aproximación pueden considerarse ambos iguales a 1.

El cálculo de *cf* es sencillo a partir del grafo del programa, es igual al número de arcos que este contiene.

Para estimar *df* hay que seguir un proceso más complicado que se detalla a continuación:

El cálculo de *cf*tiene como base el concepto de variable localmente expuesta. Esta variable está definida para un segmento de código, en este caso cada uno de los nodos del grafo correspondiente. Son las variables cuyo valor es utilizado en ese segmento (ya sea en una asignación con otra variable, en una sentencia de salida, etc.), pero que lo han adquirido en otro anterior (en una asignación o sentencia de entrada por ejemplo), *df* se calcula entonces a partir de los números de posibles adquisiciones de valor que han podido tener las variables localmente expuestas de cada uno de los módulos (Sáez, 2009).

1.3.1.4 Métricas de tamaño

Una estimación del proyecto es tan buena como la estimación del tamaño del trabajo que va a llevarse a cabo, el tamaño representa el primer reto del planificador de proyecto. El tamaño se refiere a una producción cuantificable del proyecto de software (Pressman, 1998).

Se puede decir que los programas más grandes son los más complejos, aunque solo sea por la cantidad de información que hay que considerar para poderlos entender. Una medida para medir la complejidad del programa viene dado por su tamaño. El problema está en qué definir como tamaño, qué es lo que hace que un programa sea grande o pequeño.

A diferencia de los elementos físicos, la medición del tamaño del software es difícil y no existe un real consenso respecto de una forma adecuada de medirlo. Medidas clásicas como LOC (líneas de código) son de dudosa utilidad ya que el esfuerzo de escribirlas varía enormemente aun dentro del mismo ambiente y mucho más cuando se cambia de lenguaje de bajo nivel a lenguajes de cuarta generación o ambientes de programación visuales.

El tamaño del software puede ser medido desde diferentes perspectivas, se puede considerar la longitud o el tamaño físico, la funcionalidad desde el punto de vista de lo que el usuario recibe y la complejidad del problema de lo que se pretende resolver.

Líneas de Código Fuente (LOC)

Una de las métricas más usada de la longitud de código es la línea de código.Se considera a la sentencia fuente lógica como línea estándar de código. Ahora bien, definir una línea de código es difícil porque existen diferencias conceptuales cuando se cuentan sentencias ejecutables y de declaraciones de datos en lenguajes diferentes. El objetivo es medir la cantidad de trabajo intelectual puesto en el desarrollo de un programa.

Las líneas de datos de código están basadas en declaraciones de instrucción (LOC lógico) e incluyen código ejecutable y definiciones de datos, pero excluyen comentarios.

Las objeciones más importantes que se levantan frente a las líneas de código provienen de su extensivo uso como sustituto de una mejor medida de complejidad, debido a la facilidad para el conteo, fijar criterios para contarlas y la sencillez de automatización del conteo.

Un problema básico es que la cantidad de LOC en un programa software es negativamente correlacionado con la eficacia de diseño. El objetivo del software es proporcionar cierta funcionalidad para solucionar algunos problemas específicos o realizar ciertas tareas. El diseño eficiente provee la funcionalidad, demuestra el verdadero esfuerzo y esto puede ser posible con menos LOCs. Por esta razón la cantidad de LOC no muestra la eficacia del diseño (Salvetto, 2006).

Para minimizar esos problemas, se usan listas de chequeo de definición desarrolladas por el SEI (Instituto de Ingeniería de Software), que permiten unificar criterios en la definición de una línea de código fuente. Existen herramientas automatizadas para medir la cantidad de líneas de código fuente, como por ejemplo Amadeus (1994). Para realizar un análisis de mayor especificidad, Amadeus automáticamente recolecta medidas adicionales como total de líneas fuente, de comentarios, declaraciones, interfaces, anidamientos, sentencias ejecutables y otras. Esta herramienta provee varias medidas de tamaño, incluyendo métricas aplicables a tecnologías de objetos de Chidamber, (1994).

En ambientes de programación orientados a objetos parece más adecuado contar clases, métodos, y no LOC. No obstante no existiendo por el momento una medida mejor sigue teniendo cierta utilidad en la medida en que se comparan sistemas desarrollados con la misma tecnología y bajo circunstancias similares.

Las LOC son un indicador que su utilización deberá ser valorada por el apoyo del juicio experto humano.

Uno de los principales problemas de esta métrica es que se dispone de ella después de haber finalizado el proceso de desarrollo. Cuando se usa como entrada un modelo de estimación, de lo que se dispone es de una estimación del tamaño esperado.

Una aplicación de software es un conjunto de líneas de código que se ejecutan en una computadora. Sin embargo, mucho del costo de producir ese software no está directamente relacionado con la codificación, que es entre el 20 y 25 % del costo total. Elementos como la administración del proyecto, el nivel de detalle de la documentación técnica o la documentación de pruebas, y las pruebas por sí mismas también deben considerarse.

Métricas de Halstead

En Halstead, (1977) se considera que el código está formado por unas unidades llamadas operadores y operandos y estos operadores y operandos contribuyen de manera distinta a la complejidad. Es necesario considerar además del número total de elementos (operando y operadores), el número de éstos que son diferentes (esto es, el vocabulario del programa). Investigando las relaciones entre estas cuentas, se pueden obtener unos cuantos parámetros que intentarán medir diferentes aspectos de la complejidad del programa.

Notación para el cálculo de la métrica

n1: número de operadores diferentes.

n2: número de operandos diferentes.

N1: número total de operadores.

N2: número total de operandos.

n: vocabulario de un programa (n = n1+ n2).

N: longitud total del programa (N = N1+ N2).

Para el cálculo del valor de N de una manera bastante aproximada Halstead propone la siguiente fórmula: $N^* = n1\log_2 n1 + n2\log_2 n2$

Se puede definir el volumen de un programa como: $V = N\log_2 n$

El volumen potencial: $V^* = N^*\log_2 n^*$

El volumen pretende ser una medida más precisa de la dificultad de entender un programa al tener en cuenta no solo su longitud N sino también su vocabulario. Y es lógico que con igual tamaño un programa con poco vocabulario sea más sencillo que uno que use mucho vocabulario.

El nivel de un programa brinda una idea del nivel de detalle con que ha sido codificado, se entiende que cuanto más código use para una función dada, de más bajo nivel es. El nivel del programa se define: $L = V^*/V$.

Con el Volumen del programa y el nivel se puede calcular la inteligencia contenida en el programa: I = LV.

Según Halstead este valor se correlaciona bastante bien con el tiempo total de programación y depuración (Sáez, 2009).

Puntos de Función (PF)

En ambientes integrados de programación visual es posible generar interfaces completas sin generar una línea de código, por lo que se hace necesario otro elemento diferente a la longitud para medir el tamaño

El estándar ISO de medida de tamaño funcional, define el tamaño funcional como: “un tamaño del software derivado mediante la cuantificación de los requerimientos funcionales del usuario”.

De manera general, la funcionalidad es más significativa a la hora de estimar que la longitud física.

Una medida de tamaño funcional son los Puntos de Función de Albrecht. La cual se describirá brevemente a continuación tomando como referencia el libro Análisis de Punto de Función de Dreger (1989).

Las métricas para puntos función están basadas en las guías proporcionadas por el "International Function Point UserGroup” (IFPUG). Los Puntos Función procuran cuantificar la funcionalidad de un sistema de software a partir de especificaciones independientemente de la tecnología que se vaya a usar para su desarrollo. La meta es obtener un número que caracterice completamente al sistema. Para obtener dichas especificaciones no se requiere de una metodología específica.

El primer paso es el cálculo de los puntos funcionales sin ajustar, para ello se necesita determinar a partir de la especificación:

Tipos de datos

- Ficheros lógicos internos (ILF)
- Ficheros de interfaces externos (ELF)

Tipos de funciones de transacciones

- Entradas externas (EI)
- Salidas externas (EO)
- Consultas externas (EQ)

Una vez identificados esos elementos se les clasifica al tener en cuenta la complejidad en alta, media o baja y se le asigna un peso. Los puntos funcionales sin ajustar (PFSA) se calculan sumando las cantidades de cada tipo de objeto, multiplicadas por la ponderación que les corresponde.

Después de calcular los PFSA se calculan los Puntos Funcionales Ajustados (PFA) multiplicándolos por un factor de ajuste que se calcula tomando en cuenta 14 factores de complejidad técnica. Estos factores son valorados en una escala de 0 a 5. La valoración de estos factores puede generar una variación de ±35 % (MacDonell, 1994).

Diversos autores han criticado esta técnica, a continuación se exponen una serie de problemas que presenta esta métrica desde la perspectiva de varios autores:

Kemerer (1993) encuentra que:

- Para realizar buenas predicciones de tiempo y esfuerzo es necesario incorporar aspectos que caracterizan el ambiente de desarrollo.

Fenton (1997) menciona:

- Subjetividad en el factor tecnología, el factor debido a la tecnología puede variar en ± 35 %.
- Uso temprano, para calcular los PF se requiere una especificación completa del sistema.
- Su cálculo no pude automatizarse completamente por depender del juicio experto
- No son independientes de la metodología de análisis y diseño.

Kitchenham y Känsälä (1997). Suponen que un método más simple podría reducir tal error de conteo.

Symon plantea (1998) que Los pesos subjetivos originales fueron tomados de la experiencia de IBM, lo que puede no ser aplicable en otros ámbitos.

Roberto Meli (1998) identifica los siguientes problemas en los puntos funcionales:

- no son suficientemente tempranos para la gestión del proyecto ya que cuando se dispone de los elementos para calcularlos se ha invertido entre el 15 % y el 40 % del esfuerzo,
- no miden el valor para el usuario del sistema resultante sino más bien la dificultad para desarrollarlo.

Esta técnica debería simplificarse de forma tal que un conteo básico pueda ser recolectado en forma automática, desde una representación temprana del sistema.

De acuerdo con una estimación temprana los puntos de función tienen el inconveniente de asumir que se conocen el conjunto de datos que será usado por el sistema, para clasificarlo según su complejidad en bajo, medio y alto. En un periodo temprano de concepción del producto no se cuenta con esa información.

Para poder establecer la cantidad de puntos función que tiene una aplicación, existen varios métodos de conteo. En general todos estos métodos establecen un conteo basado en la identificación del tipo de funciones que tiene la aplicación, y a cada una le asocia un número de puntos función tomando en cuenta su complejidad. Las variantes surgen al buscar conteos más precisos en puntos función conforme al tipo de aplicación. Por ejemplo, un sistema en tiempo real tiene una complejidad muy distinta a un sistema tradicional de negocio, o a un sistema operativo o a una aplicación científica que realiza muchos cálculos, pero el resultado puede ser un solo dato.

Los métodos que están homologados con el estándar ISO 14143, aunque no todos son públicos, son:

- ISO/IEC 20926:2003 Software engineering -- IFPUG 4.1 Unadjusted functional size measurement method. Este método ha sido definido por el International Function Point UsersGroup (IFPUG13) y evolucionado, a partir de la propuesta original de Allan Albrecht en IBM, por lo que es el más conocido y más utilizado, sobre todo en Estados Unidos que es el mercado más grande de software. Esto es muy importante porque se está convirtiendo en el estándar de la industria.
- ISO/IEC 24570. Software engineering -- NESMA -- Function Point Analysis. Estándar definido por la NEtherlands Software MetricsUsersAssociation. Esta es una pequeña variante del método del IFPUG.
- ISO/IEC 20968:2002 Software engineering -- Mk II Function Point Analysis. Este método ha sido desarrollado por la UnitedKingdom Software MetricsAssociation, simplificando el método y haciéndolo compatible con ideas de análisis y diseño estructurado.
- ISO/IEC 19761:2003 Software engineering -- COSMIC-FFP -- A functional size measurement method. Este método ha sido definido por el COmmon Software Measurement International Consortium, integrado por expertos de Australia, Canadá, Finlandia, Alemania, Irlanda, Italia Japón, Holanda y Reino Unido. La idea principal es adecuarse mejor a la medición de sistemas en tiempo real.

Estos métodos de conteo de puntos de función son un ejemplo de la evolución que ha seguido esta técnica en aras de ganar en eficiencia. Es válido señalar que en la actualidad estos problemas no han sido resueltos completamente.

Puntos de Objetos

A pesar de que la estimación a través de Puntos Objeto es un enfoque de medición de tamaño de software relativamente nuevo, es apropiado para las aplicaciones con componentes y para estimar esfuerzos en las etapas de prototipación (Luis, 2003).

Procedimiento (Moreno, 2010):

- Determinar la cantidad de objetos: estimar la cantidad de pantallas, reportes, componentes de tercera generación que contendrá la aplicación.
- Clasificar cada instancia de un objeto según sus niveles de complejidad (simple, media o difícil).
- Dar el peso a cada objeto según el nivel de complejidad. Los pesos reflejan el esfuerzo relativo requerido para implementar una instancia de ese nivel de complejidad.
- Determinar la cantidad de puntos objeto, sumando todos los pesos de las instancias de los tipos de objetos especificados.
- Estimar el porcentaje de reutilización que se espera lograr en este proyecto. Calcular los nuevos puntos objeto a desarrollar.
- NOP = puntos objetos x (100 - % Rehusabilidad)/100
- Determinar un ratio de productividad PROD = NOP/Meses-persona.
- Calcular el valor Meses-persona estimado según la ecuación MM = NOP/PROD

Puntos de Casos de Uso (UCP)

En 1993 Karner (1993) introduce el concepto UCP, basado en FP y el Proceso de Objectory. El proceso de conteo se basa en el modelo de casos de uso. Los UCP identifican sólo dos elementos: Actores y Casos de uso. La complejidad de cada elemento se clasifica como baja, media o alta. La misma se establece mediante el análisis de las características del actor, y para los casos de uso, a través del número de transacciones y el número de objetos de análisis involucrados en cada caso de uso. La tabla 1 muestra los valores de los factores de complejidad para cada elemento básico definidos por Karner.

	Alta	Media	Baja
Actor	1	2	3
Caso de Uso	5	10	15

Tabla 1. Factores de complejidad que participan en el cálculo de UCP (Karner, 1993)

Por lo tanto, los puntos casos de uso no ajustados se definen como la suma de los productos de los factores de complejidad (Wij) por el número de los elementos clasificados según su grado de complejidad (Iij). Esto puede ser expresado de la siguiente forma, formula (1):

$$UCP = \sum_{i=1}^{2} \sum_{j=1}^{3} Wij * Iij \qquad (1)$$

Los UUCP pueden ser ajustados si se tiene en cuenta la tecnología y el ambiente de la aplicación. A cada característica se le asigna un grado de influencia dentro de un rango que va desde “sin influencia” -para la cual se asigna el valor 0-, hasta “fuerte influencia” -para la cual se asigna el valor 5-. En consecuencia, el TCF se calcula según la fórmula (2) y el Factor ambiental (Evironmental Factor, EF) según la fórmula (3).

$$TCF = 0.6 + 0.01 \sum_{i=1}^{18} DIi * Wi \quad (2) \qquad EF = 1.4 - 0.03 \sum_{i=1}^{8} DIi * Wi \quad (3)$$

Entonces, el tamaño de la aplicación se define en términos de UCP, y se calcula como el producto de los UUCP por el TCF y el EF.

Tamaño funcional [UCP] = UUCP * TCF* EF.

Entre las cosas buenas del método se puede citar que trabaja bien con diferentes tipos de software, muestra buen rendimiento en proyectos pequeños, medianos y grandes. Es una nueva métrica que se adapta mejor con paradigmas más avanzados de programación e Ingeniería de Software. El método no está exento de problemas que hacen que no se haya consolidado. Entre estos problemas destacan la no existencia de un estándar para describir casos de uso, eso hace que aplicar la métrica sea más engorroso, hay un alto grado de subjetividad a la hora de calcular los factores técnicos y ambientales que hacen que el cálculo no sea del todo realista. Se puede observar que entre los autores que han usado los UCP no se distingue un criterio único aplicado para el cálculo de la complejidad de un caso de uso: por ejemplo, Anda (Anda, B.C.D., Benestad. H.C. and Hove, 2005) y Diev (Diev, 2006) no tienen en cuenta los objetos de análisis. Además, también existen diferentes criterios para identificar una transacción. Unos definen la transacción como un conjunto atómico de actividades entre el actor y el sistema, que debe ser realizado en forma completa (Anda *et al.,* 2001; Anda *et al.,* 2002; Anda *et al.,* 2005; Carroll, 2005; Diev, 2006). Otros claramente definen las diferencias entre transacciones y escenarios (Diev, 2006), mientras otros no hacen ninguna diferencia entre ellos (Anda *et al.* 2005). Hay otro que define el número de transacciones como el número de transacciones dentro de los escenarios de un caso de uso (Issa A., 2006).

La estimación por Puntos de Caso de Uso resulta muy efectiva para estimar el esfuerzo requerido en el desarrollo de los primeros casos de uso de un sistema, si se sigue una aproximación iterativa como RUP. En éste tipo de aproximación, los primeros casos de uso a desarrollar son los que ejercitan la mayor parte de la arquitectura del software y los que a su vez ayudan a mitigar los riesgos más significativos (iteraciones de Elaboración en el Proceso Unificado). Fuera de éste contexto, el método tiende a sobredimensionar el esfuerzo requerido.

Transacciones, Objetos de Entidad y Caminos

Robiolo (2009) propone un conjunto de tres unidades de medida de software que se identificarán con los nombres Transacciones (T), Objetos de Entidad (EntityObjects, EO) y Caminos (Paths, P). Cada una de ellas capturará un atributo interno del software: funcionalidad, datos y complejidad, respectivamente.

El objetivo de estas medidas, según la autora, es simplificar el método de conteo; definir métricas que sean inherentes a UML; obtener unidades de medida diferentes que, individualmente, puedan capturar un aspecto clave de las aplicaciones de software, para, de esa forma, poder reducir el error de estimación de esfuerzo y, al mismo tiempo, ser consistente con la teoría de la medición.

Para estas medidas se toma como entrada el modelo de casos de uso por ser un artefacto que está disponible en una etapa relativamente temprana. A partir del modelo se forma una lista con los casos de uso y se procede a realizar una descripción textual de cada uno de ellos, a partir de esta información se puede reconocer la interfaz gráfica, las entradas y las salidas de cada una de ellos.

Caminos (P)

Al definir la complejidad ciclomática, se dijo que el criterio de cálculo de complejidad en un grafo se puede definir como la cantidad de arcos agregados al flujo principal. Es posible aplicar este criterio a cada T de la descripción textual de un caso de uso si se tiene en cuenta lo siguiente:

- cada transacción es un grafo;
- el flujo principal de la transacción es el texto principal del caso de uso, por lo tanto, el valor de P es 1;
- cada expresión similar a “si… entonces” es considerada una decisión binaria;
- cada expresión similar a “en caso de: A… entonces, B… entonces, C…entonces” es considerada una decisión múltiple.

Teniendo en cuenta los aspectos anteriores la complejidad de un caso de uso se mide como se muestra a continuación, para obtener los P de una transacción se debe:

- Contar la cantidad de decisiones binarias.
- Identificar las decisiones múltiples, y asignar un número igual a la cantidad de opciones, menos una.
- Agregar 1 a la suma de los elementos 1 y 2.
- Sumar los P de cada transacción del caso de uso.

$$ComplejidaddelaAplicacion[P] = \sum_{i=0}^{n} (CantidaddePparaT1 + \cdots + CantidaddePparaTm)$$

Después de la demostración la autora arriba a la conclusión de que las métricas son efectivas para medir el esfuerzo con las técnicas antes mencionadas, en cada una de las métricas propuestas hay una disminución del error respecto a las ya conocidas aunque no muy significativo.

Puntos de Características

Este método fue propuesto por Caper Jones como una alternativa que permitiera obtener puntos de función en software científico y de ingeniería. Para evitar confusiones con los PF, Jones lo denominó puntos de característica. Actualmente es usado con mucho éxito en software del tipo CAD (del inglés ComputerAidedDesign), sistemas embebidos y sistemas en tiempo real. Algunos ejemplos de aplicaciones en las que el método de Albrecht no era del todo exacto, y que permitió el nacimiento de esta nueva técnica entre otras son: software de tiempo real para control de armamento en las fuerzas armadas, nuevos sistemas operativos, software embebido, software de comunicaciones (por ejemplo para control de telefonía), software de control de dispositivos.

Para ello, los puntos de características, añaden un componente adicional a los cinco (EI, EO, EQ, ILF y ELF) ya mencionados de los PF de Albrecht. Este componente denominado algoritmos complejos (CA – del inglés complexalgorithms) permite introducir en el conteo a procesos con algoritmos complejos o lógica de nivel elevado.

Las diferencias entre los PF de Albrecht y los puntos de características, son básicamente que cuando el número de algoritmos de una aplicación es mucho mayor que el de los archivos, los puntos de características generan estimaciones más precisas que los FP. A la inversa si existe un número de archivos considerablemente mayor al de algoritmos (situación muy común en los sistemas de gestión), la cuenta de puntos de características será mucho menos confiable que la de PF (Jones, 1984).

Mark II

MK II FPA: propuesto por Charles R. Symons (1998), este método es una derivación de los PF de Albrecht, el que considera al sistema que se está analizando, compuesto por cinco tipos de componentes (entradas externas, salidas externas, consultas externas, grupos de datos lógicos internos y externos), mientras que el MK II FPA mira al sistema como una colección de transacciones lógicas discretas, compuestas cada una de ellas por entrada, proceso y salida. Si se usan herramientas modernas de diseño para el desarrollo del software, y esas herramientas permiten identificar fácilmente las transacciones lógicas, resulta apropiado el uso de este método.

El método se puede usar preferentemente en software de gestión; pues las reglas y los valores están orientados a este tipo de software. Sin embargo, si se plantea la necesidad de medir otros tipos de software como científico y, o de ingeniería, los cuales incluyen algoritmos complejos, se podrían adecuar dichas reglas para llevar a cabo mediciones en estos dominios.

Una de las primeras diferencias que se puede apreciar con el método de PF de Albrecht, es que mientras este considera cinco componentes básicos (EI, EO, EQ, ILF y ELF), MK II examina al sistema desde el punto de vista de las transacciones lógicas, conformadas cada una por una combinación de entrada, proceso y salida, desencadenada por un evento de interés para el usuario, o una necesidad de recuperar información.

Nótese que a diferencia del método de Albrecht, MK II achica el rango del ajuste de complejidad técnica.

3D – FUNCTION POINTS

La mayoría de las técnicas de estimación, especialmente la de PF de Albrecht, tuvieron sus inicios cuando el desarrollo del software estaba orientado hacia los procedimientos y los datos. Mientras las técnicas denominadas tradicionales dividen el espacio de soluciones en datos y procedimientos, el paradigma de diseño orientado a objetos hace una combinación de ellos (Whitmire, 1992).

El problema de las técnicas que no son orientadas a objetos, es que solamente miden funcionalidad (una sola dimensión). Se puede decir, que los métodos orientados hacia la funcionalidad, ignoran aspectos importantes como son las relaciones entre objetos y la reusabilidad de estos. La funcionalidad (es decir el comportamiento de los objetos) puede resultar importante a la hora de dimensionar el esfuerzo; sin embargo, considerar esto solo puede resultar insuficiente para estimar el tamaño del software.

Esta tecnología fue desarrollada para dar respuesta a desarrollos denominados independientes de la tecnología y poder así obtener métricas convenientes para un mayor rango de dominios.

El método 3D se basa en que la aplicación a desarrollar se puede expresar en tres dimensiones: datos, funciones y el control.

Cada dimensión representa una parte de toda la complejidad de la aplicación. Si bien es cierto, se puede identificar una dimensión con un grado de complejidad mayor que otra, el análisis que se debe hacer, debe abarcar a las tres dimensiones si se quieren obtener medidas más confiables en estos tipos de software.

La dimensión de datos, puede asociarse directamente con el tipo de software denominado de gestión. En estos casos las características del software a desarrollar pueden obtenerse directamente de los manuales de prácticas de los FP de la IFPUG.

La dimensión de funciones está relacionada con el tipo de software denominado científico donde se identifica una cantidad importante de cálculos y algoritmos complejos.
La dimensión de control está fuertemente asociada con tipos de desarrollo para software denominado de tiempo real.
El autor del método ha hecho aportes importantes al desarrollo de métricas para análisis y programación orientados a objetos. Debido a que la técnica fue concebida en el seno de una compañía internacional (Boeing Company), no ha habido difusión de literatura como en otros métodos, quizás por la personalización que allí se hizo.

PUNTOS COMPLETOS DE FUNCIÓN (FFP)

Full Function Points (Puntos de Función Completos): esta técnica ha sido desarrollada por un equipo de la Universidad de Québec en Montreal (Canadá) [Abran A., et al 98], siendo muy eficiente en la medición de puntos de función en sistemas de control, tiempo real y embebidos. Particularmente, los sistemas en tiempo real presentan dos factores críticos: primero, el tiempo de respuesta y en segundo lugar, su interacción con entidades externas. Los FP de Albrecht eran limitados a la hora de medir sistemas software en tiempo real. Estos últimos trabajan con dos tipos de estructura de datos de control: grupo lógico de ocurrencia múltiple, que puede tener más de una instancia por cada tipo de registro y el grupo lógico de ocurrencia única, que puede tener solamente una instancia por cada tipo de registro. En lo que respecta a las transacciones, los sistemas en tiempo real, presentan una variación importante en la cantidad de subprocesos por proceso; es decir, el método debería contemplar que unos procesos contarían con unos pocos subprocesos y otros en cambio, con un número significativo de ellos. Los FP se basan más en el número de procesos que en el de subprocesos. Para paliar este inconveniente, FFP introduce en el cálculo no solamente a los procesos, sino que también incluye a los subprocesos.

COSMIC-FFP

Como se referencia en (Ovejero, 2006) este método ha sido inspirado en los puntos de función completos (FFP – del inglés full functionpoints) y es el resultado de diversas pruebas de campo y opiniones recibidas de distintas empresas que lo han usado en procesos de desarrollo y, o mantenimiento de productos de software. El método FFP, fue propuesto por primera vez en el año 1997 con el objeto de obtener medidas para software de tiempo real. Diversas pruebas han demostrado además, que el método es útil para software técnico, de explotación y de gestión. COSMIC FFP no es otra cosa que el Common Software Measurement International Consortium o Consorcio Internacional Común de Medidas del Software, con la participación de

varios países entre ellos Canadá, Reino Unido, Francia y Japón. Básicamente apunta a satisfacer las necesidades de:

- los proveedores de software que deben traducir los requerimientos del cliente en un tamaño del software como un paso fundamental para la estimación de los costos del proyecto.
- los clientes que deben conocer ese tamaño funcional del software recibido como un componente de la medición del desempeño del proveedor.

Esta técnica considera a los requerimientos desde una perspectiva del usuario. Estos requerimientos y en el marco de la técnica COSMIC-FFP, reciben el nombre de requerimientos funcionales del usuario (FUR - del inglés functionaluserrequirement). El modelo del proceso de medición, se muestra en la figura 2. Para aclarar algunos puntos que se han visto y se verán más adelante, se definen los siguientes conceptos relacionados con el método:

- Capa: el software que se quiere medir puede estar dividido en dos o más partes, cada una de ellas con diferentes niveles de abstracción. Cada una de estas partes recibe el nombre de capa y encapsula funcionalidades útiles en sí misma. Puede haber relación jerárquica para el intercambio de datos entre capas a nivel subordinado o entre pares (peer to peer).
- Frontera (o límites): es un conjunto de criterios percibidos mediante los FURs del software, que permite establecer una distinción conceptual clara entre aquellos ítems que son parte del software (o que están dentro de los límites) y los que forman parte del medio ambiente (están fuera de los límites del sistema).
- Usuario del software: el modelo COSMIC-FFP define al usuario como aquella persona o dispositivo de ingeniería que se beneficia con el uso del software.
- Requerimientos funcionales del usuario: son las partes de los requerimientos del software que describen la naturaleza de las funciones a ser provistas. Aquí no se incluyen a aquellos requerimientos técnicos o de calidad que igualmente describen funcionalidad del software.

Según lo ilustra la figura 2, previo a la aplicación de reglas y procedimientos de medición, se deben representar los FURs en un modelo específico que capture sus conceptos, definiciones y relaciones. Una vez efectuada esta representación, se realiza la medición aplicando el conjunto de reglas y principios del método

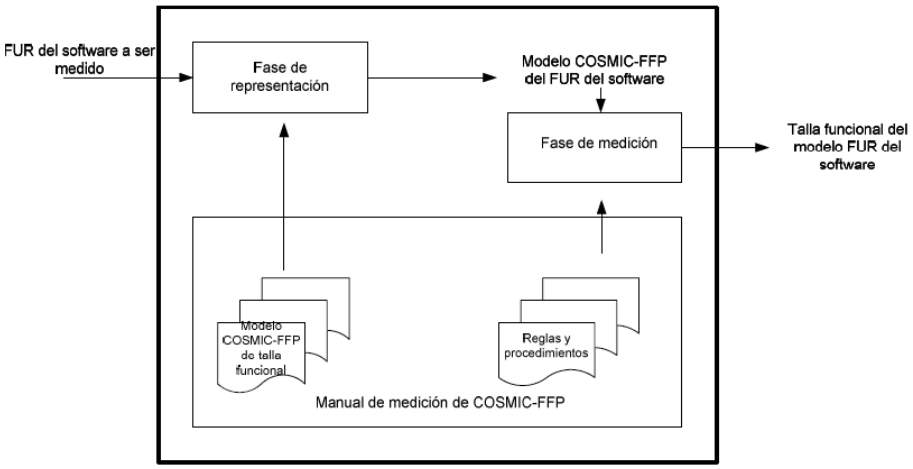

Fig2. **Modelo del proceso de medición COSMIC-FFP (Ovejero, 2006)**

Para evaluar la complejidad a pesar de las métricas antes mencionadas se pueden hacer uso de algunos métodos de estimación, que nos ofrecen factores como esfuerzo, tiempo, y costos los cuales pueden dar ideas de la complejidad de un producto informático algunos de estos métodos se tratan a continuación.

SLIM. Ecuación del Software

La ecuación del software [Norman E. Fenton'91] es un modelo multivariable que asume una distribución específica del esfuerzo a lo largo de la vida de un proyecto de desarrollo de software. El uso fundamental de la ecuación del software es para estimar la producción de software, el tiempo de desarrollo y el esfuerzo, al comienzo de un nuevo proyecto, en este punto el índice de productividad, medida de la eficiencia del desarrollo de software en un proyecto u organización, es conocido habiendo sido calculado en base a los proyectos anteriores. Estimar tan tempranamente resulta una tarea bastante compleja a continuación se presentaran algunos métodos que harán que esta predicción sea algo más confiable.

La ecuación del software tiene esta forma:

$$\text{Producto} = PP * \left(\text{ESfuerzo}/B\right)^{1/3} * (\text{Tiempo})^{4/3}$$ Donde:

Producto: Numero de líneas de código fuente desarrolladas.

PP: Parámetro de productividad, obtenido mediante calculo en base de proyectos ya finalizados. El parámetro de productividad refleja:

Madurez global del proceso y de las prácticas de administración.

La amplitud hasta donde se utilizan correctamente las normas de la ingeniería del software.

El nivel de los lenguajes de programación utilizados.

Las habilidades y la experiencia del equipo del software.

La complejidad de la aplicación.

Esfuerzo: son los anos-Hombres de trabajo necesarios para completar todo el proceso de desarrollo.

B: Es un factor especial que es función del tamaño del producto. Crece lentamente conforme el tamaño supera a las 18000 líneas de código (SLOC) hasta las 100000 debido a la gestión global del proyecto.

Tiempo (en años) corresponde al que se emplea durante el proceso de desarrollo.

Teniendo en cuenta los parámetros involucrados en la ecuación del software dos de ellos son una incógnita para un nuevo proyecto, el tiempo y el esfuerzo. El factor de productividad se obtiene de proyectos anteriores entre tanto B es un factos que depende del tamaño del producto. Para las dos incógnitas que quedan en la ecuación existen tres métodos para su obtención:

- Solucióndeterminista.
- Porsimulación.
- Por Programación Lineal.

Solución determinista

Esta solución involucra dos ecuaciones ya que el número de incógnitas debe ser igual al número de ecuaciones para poder arribar a una solución determinada. Una de las ecuaciones es la ecuación del software, la otra ecuación está basada en la tasa de incorporación del personal al proyecto y se describe de la siguiente manera:

$$\text{ParámetrodeIncorporación} = {}^{(\text{EsfurzoTotal})}/_{\text{Tiempo}^3}$$

En el proyecto completado tanto el tiempo como el esfuerzo total son conocidos, por lo que el Parámetro de Incorporación del personal puede ser calculado mediante la calibración de proyectos anteriores ya completados.

Ya al disponer de una estimación de las SLOC, se conocen los parámetros de productividad y el parámetro de incorporación las ecuaciones pueden ser resueltas simultáneamente.

Simulación

La solución determinista parte de que el estimador conoce ciertas cantidades de forma exacta pero la verdad es que no es posible trabajar con tanta exactitud ante la llegada de un nuevo proyecto. Una solución más aceptable es tratar las incertidumbres asociadas con la información de entrada como parte del proceso de obtención de la predicción. Una forma de incorporar dichas incertidumbre se consigue con la simulación de Monte-Carlo.

El número de líneas de código durante el periodo en el que el tiempo y el esfuerzo se están estimando es en sí una estimación. Tiene un valor esperado y una desviación. De la misma forma el parámetro de incorporación para una organización y un determinado trabajo tiene un valor esperado y una incertidumbre representada por la desviación estándar.

El método de Monte Carlo deja que ambas entradas varíen dentro de los rangos permitidos por su desviación a partir del valor esperado. Se utilizan las mismas ecuaciones que para la solución determinista y se realiza de 100 a 1000 computaciones de las incógnitas de forma que se obtiene un buen número de soluciones. La media (o valor esperado) y la desviación estándar son obtenidas para ambos valores de tiempo y esfuerzo para todo ese conjunto de soluciones.

Este método es aconsejable no solo porque da un valor esperado para esfuerzo y tiempo, sino porque también da una medida de incertidumbre de dicha solución. Esta incertidumbre será una guía importante a la hora de realizar análisis de riesgos.

Programación Lineal

Existen varias restricciones que debe tomar un quipo de desarrollo al menos cinco son indispensables:

Tasa máxima de incorporación de los recursos humanos.

Punto máximo del personal empleado en el proyecto.

Mínimo punto de utilización de personal.

Tiempo contratado para la entrega del producto.

Cantidad de dinero presupuestada para el desarrollo.

Cada una de estas restricciones puede expresarse mediante ecuaciones entre cuyos términos se hallarán el esfuerzo total, el tiempo de desarrollo, esfuerzo por año y costo por año de esfuerzo. Haciendo una combinación de esas ecuaciones ya sea gráfica o analíticamente se puede encontrar un conjunto de posible soluciones.

Para utilizar este método es necesario estimar la funcionalidad del proyecto, realizar dicha estimación es bastante complejo, de hecho en muchas ocasiones ha fracasado, una posible

causa son las SLOC que se usan como medida para el tamaño, así que se han buscado otras como los Puntos de Función las cuales también tiene sus desventajas a la hora de estimar tempranamente, mencionadas en epígrafes anteriores.

COCOMO

El modelo COnstructiveCOsteMOdel fue creado en 1981 por Barry Boehm (1981). Este método es llamado usualmente como COCOMO 81, para diferenciarlo de versiones posteriores. Constituye un modelo único, este modelo asume que el tamaño de un proyecto puede ser estimado en líneas de código o puntos de función y usa ecuaciones no lineales para determinar el esfuerzo del proyecto. A continuación se presentan las ecuaciones que se deben usar para estimar el esfuerzo dependiendo del tamaño del proyecto:

Para grandes y medianos proyectos

$$\text{Esfuerzo} = a * \text{tamaño}^b$$

Para proyectos pequeños con equipos de 2 o 3 personas:

$$\text{Esfuerzo} = a * size + b$$

En ambas ecuaciones los parámetros a y b se basan en las características del proyecto que se esté tratando, para darle valores a estos parámetros se comparan esas características con la de proyectos similares.

Para aplicar este modelo lo primero que se debe hacer es seleccionar el tipo de proyecto en el que se está trabajando, Orgánico, Semi-libre o Rígido.

Luego se puede elegir el método a usar: COCOMO Básico o COCOMO Intermedio. En dependencia del método escogido serán los valores que se deben usar para su aplicación, estos valores aparecen predeterminados para el método Básico y el Intermedio. El método Intermedio requiere un poco más de esfuerzo en el momento de definir el Factor de Ajuste del Esfuerzo (EAF), el cual es multiplicado con el resto de la ecuación de esfuerzo. Para eso el estimador debe de tener en cuenta una serie de categorías del proyecto como son: atributos del producto, atributos de cómputo, atributos del personal y atributos del proyecto, hay un número de características dentro de cada una de estas categorías para las cuales se debe evaluar su impacto dentro del proyecto en una escala ascendente de 1-6, dependiendo del valor asociado al impacto hay un factor, el cual es multiplicado por los valores restantes para finalmente determinar el EAF. Existe un tercer método, el método detallado y es un refinamiento del intermedio, separa la aplicación de los manejadores de costo en Módulos, Subsistemas o diferentes Niveles del sistema.

El modelo COCOMO es un método empírico, construido en 1981, desde entonces las aplicaciones de software han variado notablemente, es por eso que el método ha evolucionado para dar respuestas a las nuevas tendencias.

COCOMO II

COCOMO se convirtió en el modelo en uno de los modelos de estimación de tiempo y esfuerzo de uso más extendido en la industria. Con el paso del tiempo este modelo fue quedando obsoleto frente a la aparición de nuevas metodologías, ciclos de vida y tecnologías. (Boehm Barry, Abts Chris, Chulani Sunita, 2000)

El mayor problema con el método analizado anteriormente es que el software se ha modernizado dando respuestas a las necesidades actuales, las tecnologías y los conceptos que sustentan dicho modelo son obsoletas. Por ejemplo el COCOMO 81 no tiene en cuenta el uso de un desarrollo iterativo. Para esto COCOMO II establece nuevos conceptos para ajustar mas la estimación con los nuevos paradigmas y características de los proyectos es por ello que define tres submodelos de estimación:

Composición de aplicaciones: para estimaciones en la primera etapa del ciclo de vida de un proyecto de software.

Diseño anticipado: preparado para sistemas en diseño, produce estimaciones tan fiables como el grado de evolución que se haya alcanzado en el diseño de la aplicación.

Post-Arquitectura: para sistemas completamente diseñados y como paso previo al comienzo de la etapa de desarrollo, así como para seguimiento del esfuerzo invertido en el desarrollo del producto.

Con respecto al COCOMO 81 también se han definido nuevos drivers de costo y se han desechado otros. La clasificación del modo de desarrollo en orgánico, semi-libre o rígido se realiza ahora dependiendo de unos factores de ajuste:

- Precedencia: experiencia en aplicaciones del mismo tipo.
- Flexibilidad de desarrollo: grado de sujeción del desarrollo a tiempo y requisitos.
- Resolución de arquitectura y riesgos: identificación de los riesgos en la aplicación.
- Cohesión del equipo: nivel de integración del equipo de desarrollo.
- Madurez del proceso: experiencia en el modelo de desarrollo.

Conclusiones

La complejidad que demanda desarrollar un producto informático esta en función de la complejidad del propio producto, evaluar esta complejidad es fundamental para estimar factores importantes dentro de cualquier proyecto como lo son los costos, el tiempo que demanda se desarrollo y el esfuerzo implicado. En el caso de los productos software aun la estimación de esos factores y la complejidad siguen constituyendo un reto.

Existe una gran variedad de técnicas y métricas que permiten evaluar la complejidad, y siendo las de tamaño las más usadas y referenciadas en el literatura consultada. Una dificultad de todas estas técnicas y métricas es que los elementos de que disponen para su evaluación están en faces avanzadas de los proyectos por lo que si un proyecto no es viable por alguna razón se incurren en gastos innecesarios.

La tendencia actualmente es trabajar con los procesos de negocios en aras de contar con elementos disponibles en la primera etapa de la concepción de un producto de software, y realizar la evaluación de la complejidad de manera que aporte una información más certera.

Referencias bibliográficas

(2009). Retrieved from The Standish Group: http://www1.standishgroup.com/newsroom/chaos_2009.php

Amadeus, A. (1994). *Measurement System User's Guide, Version 2.3a,.* Irvine, California: Amadeus Software Research, Inc.,.

Anda, B. , Dreiem, H. ,Sjoberg, D. and Jorgensen, M. (2001). Estimating Software Development Effort Based on Use Cases- Experiences from Industry. *Lectures Notes in Computer Science, Vol. 2185* , 487-502.

Anda, B., Angelvik, E. and Ribu, K. (2002). Improving Estimation Practices by Applying Use case Models. *Lecture Notes in Computer Science, Vol. 2559* , 383- 397.

Anda, B.C.D., Benestad. H.C. and Hove. (2005). A multiple-CAse study of effort estimation based on use case point. *In Fourth International Symposium on Empirical Software Engineering* (pp. 407-416). Australia: IEEE Computer Society.

Barros, O. (1994). *Reingenieria de Procesos de Negocios.*Chile: Editorial Dolmen.

Boehm Barry, Abts Chris, Chulani Sunita. (2000). Software development cost estimation approaches –. *Annals of Software Engineering* , 177-205.

Boehm, B. (1981). *Software Engineering Economics.* Prentice Hall.

Chapin, N. (1979). A measure of software complexity. *Proceedings NCC* .

Chidamber, S. C. (1994). A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering,* .

Curtis, B. M. (1992). Process modeling. *Communications ACM: 35(9)*, (pp. 75- 90).

Davenport, T. (1993). *Process Innovation.* USA: Harvard Business School Press.

de Soto, A. C. (2006). Nuevas Tendencias en Sistemas de Información: Procesos y Servicios. *Pecvnia. 2* , 129-158.

Diev, S. (2006). Use cases modeling and software estimation: applying use case points. *CAN Software Engineering Notes, Vol. 31, N. 6* , 1-4.

Doria, G. (2001). Retrieved Noviembre 28, 2011, from CIRIA UDLAP: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/gonzalez_d_h/capitulo_5.html

Fenton, N. (1997). *Software metrics : a rigorous and practical approach.* London: Chapman & Hall, 2010.

Halstead, M. (1977). *Elements of software science.* New york: Elsevier North-Holland.

Hansen, W. (1978). Measurement of program complexity by the pair(Cyclomatic number, Operator count). *ACM SIGPLAN Notices* , 29-33.

Harmon, P. (2003). *Business Process Change: A Manager's Guide to Improving, Redesigning, and Automating Processes.*

Issa A., O. M. (2006). Software Cost Estimation using Use- Case Models: a Critical Evaluation. *Information and Comunication Technologies, Vol No: 2* , 2766-2771.

Karner. (1993). *Metrics for Objectory Diploma . Diploma thesis.* University of Linkoping.

Kemerer. (1993). C.F Releability of Function Point Measurement: a field experiment. *Communication of ACM* , 85-97.

Kitchenham, B. (1997). The Problem with the Function Points. *Software IEEE. Vol 14. ISSN 0740-7459* , 29-31.

LOWENTHAL, J. (2006). *DEFINICION Y ANALISIS DE UN PROCESO DE NEGOCIO.* PANORAMA EDITORIAL ISBN: 9683813585.

Luis, O. (2003). *Métricas tricas e Indicadores.* Argentina: GIDIS, Facultad de Ingeniería, UNLPam.

MacDonell, S. (1994). Comparative review of functional complexity assessment methods of effort estimation. *Software Engineering journal* , 107-116.

McConnell, S. (2006). *Software Estimation: Demystifying the Black Art.* ISBN: 0735605351.

Meli, R. (1998). FUNCTIONAL METRICS: PROBLEMS AND POSSIBLE SOLUTIONS. *The European Software Measurement .*

Mendoza, LuisE., Perez, Maria. *PLANIFICACIÓN DE LA IMPLEMENTACIÓN.*UNIVERSIDAD SIMÓN BOLÍVAR.

Mili H., G. b. (2004). Going beyond MDA: Business Process Modeling for Software Reuse. *Workshop on Legacy Transformation: Capturing Business Knowledge from Legacy Systems - OOPSLA'2004.*Vancouver, Canadá.

Moreno, A. M. (2010). *Estimación de proyectos de software.*

Jorge Fernández Lugo, F. P. (2009, Agosto 16). Retrieved noviembre 2011, from Blog de Fernando Princich: http://flprincich.blogspot.com/2009/08/guias-practicas-de-estimacion-de.html

Ovejero, J. (2006). *TESIS DE MAGISTER EN INGENIERIA DE SOFTWARE Estimación de Proyectos para Sistemas Basados en Conocimientos.* Buenos Aires.

Pressman, R. S. (1998). *Ingeniería del software, un enfoque práctico.*España: Ed. McGraw-Hill.

(1993). *Process Innovation.* Harvard Business School Press.

Robiolo, G. (2009). *Transacciones, Objetos de Entidad y Caminos: métricas de software basadas en casos de uso que mejoran la estimación temprana de esfuerzo. tesis de Doctorado.*

Rodríguez., J. C. (2004). MODELOS SEGMENTADOS DE ESTIMACION DEL ESFUERZO DE DESARROLLO DEL SOFTWARE: UN CASO DE ESTUDIO CON LA BASE DE DATOS ISBSG. *Revista de Procesos y Métricas de las Tecnologías de la Información. VOL. 1, No:2. ISSN: 1698-2029* , 25-30.

Sáez, V. F. (2009). *Complejidad y Tecnologías de la Información.* Madrid: Fundación Rogelio Segovia para el Desarrollo de las Telecomunicaciones.

Salvetto, P. (2006). *TESIS DOCTORAL Modelos Automatizables de Estimación muy Temprana del Tiempo y el Esfuerzo de Desarrollo de Sistemas de Información. .*Universidad Politécnica de Madrid.

Selby R., A. P. (1991). Metric-Driven Analysis and Feedback systems for Enabling Empirically Guided Software Development. *Thirteenth International Conference on Software Engineering (ICSE 13)* (pp. 288-298). Austin, TX,: Proceedings of Thirteenth International Conference on Software Engineering.

Smith, H. ,. (2002). *The Emergence of Business Process Management .* CSC'S RESEARCH SERVICES.

Symons, C. R. (1998). Function Point Analisis: Difficulties and Improvements. *IEEE Transactions on Software Engineering* , 2-11.

Vernadat, F. (1996). *Enterprise Modeling and Integration: Principles and Applications.*Chapman & Hall.

Viega, L. ,. (2007). *Nota teórica Procesos de Negocio y Ventajas Competitivas.*Uruguay: Universidad ORT Uruguay.

Whitmire, S. A. (1992). Function Points: Scientific an Real-Time Extension to Function Points. *Pacific Nothwest Software Quality Conference.*

Elvira Rolón, F. R. (s.f.). APLICACIÓN DE MÉTRICAS SOFTWARE EN LA EVALUACIÓN DE MODELOS DE PROCESOS DE NEGOCIO. Tampico, Tamps. México ; Ciudad Real, España, Mexico, España.