Capítulo 3

Métricas del software

Este capítulo está organizado de la siguiente manera. En la sección 3.1 se introducen algunos conceptos acerca de la medición. En la sección 3.2 se describen las métricas del software y sus beneficios. En las secciones 3.3 y 3.4 se presentan la clasificación de las medidas del software y los diferentes enfoques de la medición. En la sección 3.5 se analizan las desventajas de las líneas de código y en la sección 3.6 se comparan con los FP. En la sección 3.7 se describen en detalle los métodos de FPA.

3.1. Las medidas: qué son y por qué hacerlas

Nuestra vida cotidiana está permanentemente rodeada de situaciones que dependen de mediciones: determinar precio, tamaño y cantidad de objetos, establecer los parámetros que requiere un avión para sus diferentes operaciones en despegue, vuelo y aterrizaje, decidir según ciertos valores el estado de salud o enfermedad de una persona o ajustar instrumental para realizar prácticas de distintos tipos en cualquier actividad científica. Éstos y muchísimos más son ejemplos de cómo inciden las medidas en nuestras vidas.

¿Por qué medir? [IFPUG'00ª]

La medición es una práctica de administración probada en el tiempo, porque:

- No se puede administrar lo que no se puede medir.
- Aproximadamente el 40% de los proyectos fracasan debido a la falta de control en la administración.
- La medición también brinda una herramienta para comunicar a los clientes el tamaño de su petición y extrapolar productividad, calidad y estimación.
- Se mide para entender y mejorar los procesos.

En este punto corresponde establecer el significado de las palabras medir, medida y medición. El *Diccionario Kapelusz de la lengua española* [Kapelusz'79] los define:

Medir: Comparar una magnitud con otra de su misma clase, tomada como unidad.

Medición: Acción y efecto de medir.

Medida: Acción y efecto de medir. Expresión numérica del resultado de una medición.

En [Fenton'96] se define formalmente de la siguiente manera:

Medición: proceso de asignar números o símbolos a los atributos de las entidades del mundo real de manera de describirlos de acuerdo a un conjunto de reglas claramente definidas.

Por lo tanto, una medición captura información de atributos de entidades, entendiendo por entidad un objeto o evento del mundo real. Para ello es necesario

distinguir los objetos según ciertas propiedades que los identifican. Esas propiedades o características de las entidades se conocen como atributos. En nuestra vida cotidiana hablamos indistintamente de medir entidades y atributos, pero, si bien es aceptable en el ámbito informal, esa forma de expresión es errónea y no válida para la ciencia. Estrictamente no se miden entidades, se miden atributos de entidades. Esos atributos se expresan mediante números o símbolos. Se representa el precio de un objeto como un número que significa la cantidad de pesos, la medida de la vestimenta mediante un símbolo (L, M, S) o un número que indica el talle o la dificultad para resolver un problema con un símbolo (fácil, medio, difícil). Esos números y símbolos son de gran importancia: nos permiten establecer comparaciones, obtener conclusiones y tomar decisiones acerca de los atributos de las entidades [Fenton'96].

En este campo se plantean una serie de interrogantes como: ¿Se pueden medir todos los atributos? ¿De qué depende la precisión de una medida? ¿Cuál es la escala adecuada? ¿Qué acciones se pueden aplicar a los resultados de la medición? ¿Cuál es el margen de error aceptable? [Fenton'96]

Para responder a esas preguntas, primero debe establecerse qué clases de cosas pueden medirse y aquí es interesante observar que en muchas ciencias en la actualidad se pueden medir atributos que en otros tiempos se creyeron no mensurables [Fenton'96].

En el campo de las mediciones existen dos clases de cuantificación: medición y cálculo. La medición implica la cuantificación directa, por ejemplo, medir la estatura de una persona. El cálculo es una forma de cuantificación indirecta, es decir, la medida se obtiene a partir de tomar medidas y combinarlas para obtener el valor de un atributo de interés, por ejemplo, medir la velocidad de movimiento de un objeto usando las medidas de distancia y tiempo. Esas dos formas de cuantificación se conocen como medidas *directas* e *indirectas* [Fenton'96].

3.2. Las métricas del software

La medición del software se ha convertido en parte esencial de la Ingeniería del Software. Muchos de los desarrolladores de software miden características del software para saber si los requerimientos son consistentes y completos, si el diseño es de alta calidad y si el código está listo para ser probado. Los administradores de proyectos miden atributos de procesos y productos para ser capaces de decir cuando el software estará en condiciones de ser entregado y si el presupuesto estará excedido [Fenton'96].

La Ingeniería del Software describe la colección de técnicas que aplican un enfoque ingenieril a la construcción y soporte de productos de software. Las actividades de la Ingeniería del Software incluyen la administración, cálculo de costos, planificación, modelado, análisis, especificación, diseño, implementación, prueba y mantenimiento. Con la expresión enfoque ingenieril, se quiere significar que cada actividad es comprendida y controlada. La importancia de la Ingeniería del Software no se puede subestimar, debido a que los productos de software invaden nuestras vidas [Fenton'96].

Las disciplinas de la ingeniería se basan en modelos y teorías, aplican métodos científicos sobre la base de hipótesis, diseñan y realizan experimentos para probar su verdad y analizan sus resultados. La medición representa el soporte para el proceso científico. La utilidad y efectividad de la ciencia y la ingeniería dependen fuertemente de las mediciones. Así como nadie tiene dudas acerca del rol principal de las mediciones en las diferentes ramas de la ingeniería, durante mucho tiempo fueron consideradas un lujo en el ámbito de la Ingeniería del Software y cuando se hicieron, a menudo fueron inconsistentes e incompletas. Está muy claro desde las otras disciplinas de la ingeniería que las mediciones pueden ser efectivas, si no esenciales, para valorar la dimensión y decidir soluciones a los problemas [Fenton'96].

La medición debe considerarse una actividad científica motivada por objetivos claros, comprensibles y ajustados a las necesidades de usuarios, desarrolladores y administradores.

Las métricas del software incluyen muchas actividades, que comprenden estimaciones de costo y esfuerzo, medidas de productividad, recolección de datos, medidas de calidad y confiabilidad, evaluación de performance, métricas de complejidad, evaluación de métodos y herramientas [Fenton'96].

Las mediciones son necesarias para valorar el estado de un proyecto, de un proceso, producto o recurso. Es esencial medir y registrar características, tanto de buenos como de malos proyectos. Necesitamos documentar las tendencias, la magnitud de las acciones correctivas y de los cambios resultantes. Tom DeMarco, un gran defensor de las mediciones, afirmó en 1982: "No se puede controlar lo que no se puede medir" [Fenton'96].

Debido a la gran importancia económica de la industria del software y a la necesidad de disponer de medidas de performance y de métodos de estimación confiables, los ingenieros del software se han visto forzados a pensar en nuevos enfoques con el fin de lograr esos objetivos. Durante muchos años se pensó que los programas de mediciones ocupan demasiado tiempo y que hay otras prioridades; con el correr del tiempo y los problemas surgidos con los sistemas de software, se comenzó a valorar la importancia de introducir programas de medición en las organizaciones, tratando que los mismos sean una actividad complementaria de los procesos de desarrollo y mantenimiento. Un beneficio de introducir programas de medición es que la acumulación de medidas de performance a largo plazo, puede ser extremadamente valiosa para ayudar a mejorar la estimación de proyectos. Producir estimaciones confiables es uno de los ingredientes claves para un mejor control del proyecto [Symons'91].

En el párrafo anterior se ha mencionado el término *performance*, es importante tener claro cuál es su significado. Hay tres aspectos a considerar con relación a la performance del desarrollo y mantenimiento, dependiendo de los objetivos. Un primer objetivo puede ser mejorar la *productividad*, es decir producir más para un cierto recurso dado o producir más con menor costo, el cual en términos de desarrollo de sistemas significa menos esfuerzo u horas trabajadas. La productividad se define como [Symons'91]:

Productividad = salida / entrada = tamaño del sistema / horas trabajadas

Otro objetivo vinculado a la performance está relacionado con la *entrega del sistema* más rápido, para esto la medida apropiada de la performance es entrega:

Entrega = tamaño del sistema / semanas consumidas

El tercer aspecto de la performance es el mejoramiento de la *calidad*. La calidad es difícil de definir y hay varias medidas posibles. Las más importantes son:

Densidad de defectos = cantidad de defectos / tamaño del sistema

También es pertinente hablar de calidad funcional, es decir la visión del cliente acerca de la calidad, en términos de facilidad de uso, confiabilidad, seguridad, precisión o de calidad técnica, es decir, la visión del proveedor acerca de la calidad del sistema, en términos de su diseño, mantenibilidad y operabilidad [Symons'91].

Cualquiera sea el objetivo planteado, en todos ellos el tamaño del sistema es una medida clave.

Si se piensa acerca del problema de estimar el tamaño total de la tarea de desarrollar un sistema, medido en horas de trabajo, entonces el tamaño tiene tres componentes [Symons'91]:

- Alguna medida de la cantidad de información a procesar por el sistema: entradas, salidas, etc.
- Alguna medida del tamaño de los requerimientos técnicos: batch vs. on-line, facilidad de uso, etc.
- Los controladores de performance (performance-drivers). Este grupo comprende todos aquellos aspectos que no son parte de los requerimientos funcionales y técnicos: administración del proyecto, objetivos de calidad, métodos, herramientas, lenguajes de programación, etc.

3.2.1. Beneficios de la aplicación de métricas

La aplicación de mediciones es muy importante porque favorece tres actividades básicas [Fenton'96]:

1. Las mediciones ayudan a *entender* qué está ocurriendo durante el desarrollo y mantenimiento. La evaluación de la situación actual permite establecer lineamientos que ayudan a fijar objetivos para futuros comportamientos. En este sentido las mediciones hacen visibles los aspectos de productos y procesos y por lo tanto mejoran la comprensión de las relaciones entre actividades y las entidades que ellos afectan.
2. Las mediciones permiten *controlar* lo que está ocurriendo en un proyecto. En base a los lineamientos, objetivos y comprensión de las relaciones se puede predecir qué puede ocurrir y realizar los cambios a procesos y productos que ayuden a alcanzar los objetivos.

3. Las mediciones estimulan a *mejorar* los procesos y productos. Se puede aumentar la cantidad o tipo de revisiones del diseño basándose en las medidas de calidad de la especificación y las predicciones de calidad del diseño.

3.2.2. Medición del tamaño del sistema

La Especificación de Requerimientos del Software es el primer producto tangible de la mayoría de los ciclos de vida del desarrollo y una de las principales fuentes de problemas en etapas posteriores. Aunque la mayoría de los atributos de calidad de los requerimientos documentados son subjetivos, hay aspectos de la documentación que se pueden medir, que son indicadores de atributos relacionados a la calidad. El tamaño es uno de los primeros usados en muchas métricas de calidad y se puede medir directamente [Wilson'97].

Cuando se va a estimar el tamaño de la tarea de desarrollo, tenemos que tener en cuenta los tres componentes mencionados anteriormente, mientras que para las mediciones de productividad, el tamaño del sistema que necesitamos es función solamente de los dos primeros [Symons'91].

La Figura 11 representa los dos componentes a considerar:

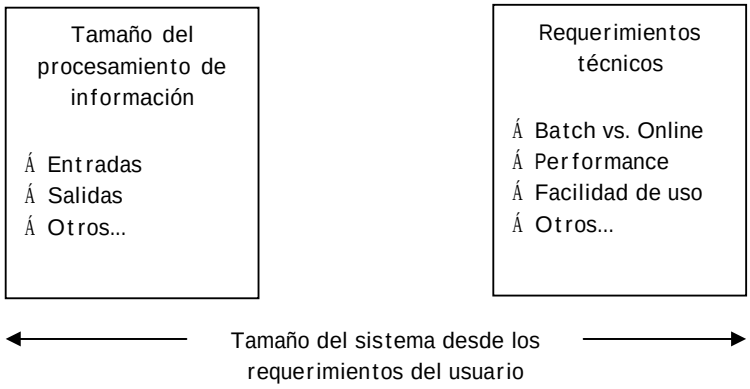

Figura 11 - Los componentes del tamaño del sistema [Symons'91]

3.3. Clasificación de las medidas del software

Teniendo conciencia de la necesidad de las mediciones, la primera tarea consiste en identificar las entidades y atributos que se pueden medir. Hay tres tipos de entidades [Fenton'96]:

Proceso: una colección de actividades relacionadas al software.

Producto: cualquier artefacto que resulta de una actividad de proceso.

Recurso: entidad requerida por una actividad de proceso.

Dentro de cada entidad se distinguen:

Atributos internos: son aquéllos que se pueden medir en términos de la entidad en sí misma.

Atributos externos: son aquéllos que solamente se pueden medir con respecto a cómo se relaciona la entidad con su entorno, o sea, su comportamiento.

Los atributos internos se pueden medir simplemente examinando la entidad, por ejemplo, el tamaño de un módulo de código o su complejidad, en tanto los externos sólo pueden evaluarse cuando el módulo se ejecuta, como es el caso de la cantidad de fallas o el tiempo que toma realizar una búsqueda y retorno de información. En general es más difícil medir atributos externos que internos y suelen medirse más tarde en el proceso de desarrollo. Puede complicarse aún más el panorama cuando esos atributos externos que se desean medir se pueden definir en términos de otros atributos que son más fáciles de medir en forma directa y no hay acuerdo en cuanto a cuáles son esos atributos, tal es el caso de evaluar el atributo externo calidad del software. Por lo tanto, es esencial identificar correctamente las relaciones entre los atributos internos y externos, así como encontrar métodos para medir en forma directa aquellos atributos de interés. La importancia de la medición de atributos internos para predecir atributos externos durante el proceso de desarrollo, se basa en la necesidad de monitorear, controlar y mejorar el proceso [Fenton'96].

3.4. Determinar qué medir

Una vez que se han identificado las entidades y los atributos, se debe determinar cuál es el objetivo de la medición. Establecer el objetivo ayudará a decidir qué entidades y atributos de las mismas deben medirse.

Debe determinarse en cada caso qué entidades son relevantes, qué atributos se van a medir y si la medición se utilizará para valoración o predicción.

Como se mencionó en el Capítulo 1, los enfoques tradicionales se han centrado en la medición del tamaño expresado en cantidad de líneas de código. Sin embargo existen otras formas de medir el tamaño de cualquier producto del desarrollo. Hay una cantidad de importantes aspectos específicos del tamaño, denominados longitud, funcionalidad, complejidad y también se puede considerar el reuso. La longitud es el tamaño físico del producto, la funcionalidad mide las funciones suministradas por el producto al usuario y la complejidad puede ser interpretada en diversas formas: complejidad del problema, complejidad algorítmica, complejidad estructural y complejidad cognitiva. El reuso mide cuánto del producto fue copiado o modificado desde otro producto existente [Fenton'96].

En este trabajo se realizarán mediciones sobre las entidades obtenidas como producto del proceso de elicitación, denominadas LEL y Escenarios. El atributo que nos interesa medir es un atributo interno, el tamaño del sistema en términos de su funcionalidad.

3.5. Líneas de código fuente (SLOC)

La cantidad de SLOC es una medida del tamaño de los requerimientos del usuario después que se ha construido el sistema y los requerimientos se han traducido a expresiones lógicas en la sintaxis de un lenguaje de programación. La gran ventaja es que las SLOC se pueden medir objetiva y automáticamente, pero tiene varias desventajas [Symons'91].

La primera de ellas y la principal es la definición de reglas para contar las SLOC en un lenguaje de programación dado y cómo sumarlas para un sistema escrito en más de un lenguaje. Este problema de la definición de reglas, que a primera vista pareciera simple, es bastante complejo, debido a que la cantidad de SLOC de un mismo lenguaje puede variar mucho según las convenciones adoptadas. Entre las convenciones que se deben establecer, pueden incluirse [Symons'91] [Fenton'96]:

- ¿Contar líneas de código procedural y/o sentencias declarativas?
- ¿Contar líneas en blanco?
- ¿Contar líneas de comentarios?
- ¿Contar las sentencias de declaración de datos solamente una vez o por cada módulo en que están incluidas?
- ¿Contar sentencias de expansión de macros y/o contar las expansiones de esas macros?
- ¿Contar las SLOC de funciones auxiliares necesarias para el procesamiento, que no son parte de los requerimientos del usuario?
- ¿Contar las SLOC de las funciones de backup y recuperación?

El problema de las convenciones es aún más complicado cuando el sistema utiliza más de un lenguaje. Se han establecido equivalencias entre distintos lenguajes, pero no hay convenciones aceptadas universalmente acerca de las mismas. Dentro de una misma organización es posible establecer e imponer algún conjunto de convenciones, pero esto se complica en general cuando se utilizan repositorios, herramientas CASE, etc. y se vuelve difícil determinar el significado de las SLOC [Symons'91].

Otra desventaja importante de SLOC es que no pueden medirse hasta que el sistema está completamente desarrollado, lo que implica que cualquier método de estimación que requiera el tamaño del sistema al principio del desarrollo se ve obstaculizado. Solamente aquellos profesionales con gran experiencia y habilidad para estimar por analogía con otros proyectos podrán utilizar esos métodos de manera confiable [Symons'91].

De cualquier manera, si se deben desarrollar sistemas en tiempo real, sistemas de control, sistemas operativos, herramientas de software, para los que algunos métodos de Análisis de Puntos Función (FPA) no son aplicables, las mediciones utilizando SLOC pueden ser el único enfoque disponible [Symons'91].

3.6. SLOC vs. Puntos Función

La cultura de muchas organizaciones de software está más cómoda haciendo estimaciones ad-hoc basándose en la experiencia previa o contando las SLOC nuevas o modificadas. Se han desarrollado muchas herramientas para contar o predecir la cantidad de SLOC, pero aún con los métodos más sofisticados, las estimaciones tempranas del tamaño y esfuerzo de los proyectos usando SLOC, especialmente aquéllas que implican un nuevo lenguaje o metodología, son inadecuadas y a menudo llevan a un exceso en costos y tiempo. Los Puntos Función (FP) son una unidad de medida lógica de las funciones del sistema de software desde la visión del usuario. Éstos proveen el valor esencial de lo que es el software y qué hace con los datos desde el punto de vista del usuario. Su potencia proviene del énfasis sobre el punto de vista externo. Debido a que la estimación de esfuerzo y costo basada en FPA no depende del lenguaje y tecnología utilizados, se puede disponer del cálculo desde las primeras etapas del desarrollo, particularmente los FP pueden calcularse desde los requerimientos. Solamente esto último ha promovido la amplia aceptación de esta técnica [Bernstein'95].

Las métricas orientadas al tamaño son objeto de polémicas y no están aceptadas universalmente como el mejor modo de medir el proceso de desarrollo de software. La mayor parte de la discusión gira en torno al uso de las SLOC como medida clave. Sus defensores afirman que es un artificio que se puede calcular fácilmente para todos los proyectos de desarrollo de software, que muchos modelos de estimación del software existentes las utilizan como elemento de entrada y que existe un amplio conjunto de datos y de literatura basados en SLOC. En el extremo opuesto, los detractores afirman que las medidas en SLOC son dependientes del lenguaje de programación, que perjudica a los programas más cortos pero bien diseñados, que no se pueden adaptar fácilmente a lenguajes no procedimentales y que su utilización en la estimación requiere un nivel de detalle que puede ser difícil de conseguir (puede ser necesario estimar las SLOC a producir antes de completar el análisis y diseño) [Pressman'93].

Muchos ingenieros de software prefieren estimar la funcionalidad en vez del tamaño físico. El concepto de funcionalidad captura la noción de cantidad de función contenida en un producto entregado o en la descripción de lo que se supone será el producto [Fenton'96]. Las métricas orientadas a la función son medidas indirectas del software y del proceso por el cual se desarrolla. Estas métricas se centran en la "funcionalidad" o "utilidad" del programa [Pressman'93].

Para sustentar este enfoque de funcionalidad se han desarrollado métodos que han sido probados y revisados sobre la base de la experiencia. Estos métodos se conocen como Métodos de Análisis de Puntos Función [Fenton'96].

Una cuestión muy importante es el ámbito de aplicación de las métricas. La medida de FP se diseñó para ser utilizada en aplicaciones de sistemas de información de gestión [Pressman'93]. Hay ciertos tipos de software en que las medidas en FP como están definidas por algunos métodos no son confiables (ver Figura 12) [Symons'91].

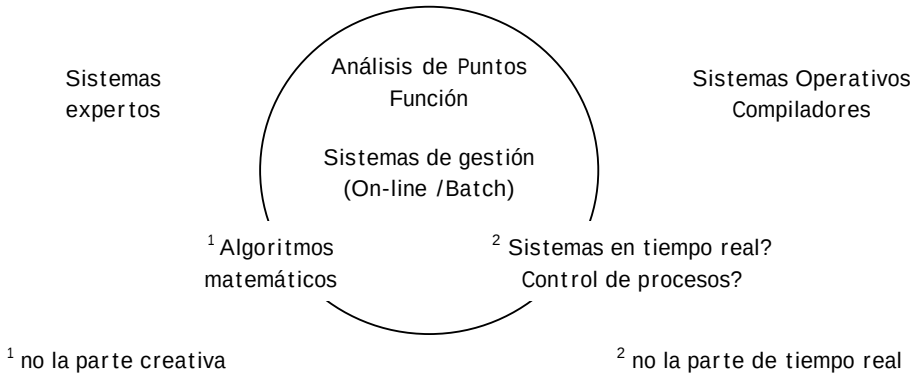

Figura 12 – Ámbito de aplicación de los métodos de FPA [Symons'91]

En la actualidad esta restricción ha sido superada por el método COSMIC FFP que puede aplicarse también a software para aplicaciones en tiempo real [Symons'01].

3.7. Métodos de Análisis de Puntos Función

3.7.1. Introducción

La definición estándar de productividad es "bienes o servicios producidos por unidad de trabajo o costo". Hasta 1979, cuando Albrecht de IBM publicó su métrica de FP no había una definición de exactamente qué bienes o servicios eran el resultado de un proyecto de software. La métrica previa era "costo por línea de código", la cual no se relaciona completamente a la definición económica de productividad. Los administradores de fábricas entienden que si un proceso de fabricación implica un porcentaje sustancial de costos fijos y disminuye la cantidad de unidades fabricadas, el costo por unidad debe aumentar. El software implica un porcentaje sustancial de costos fijos que no están asociados con el código. Cuanto más poderosos son los lenguajes de programación que se usan, el resultado es la reducción de las unidades que se deben producir para un sistema. Sin embargo, los requerimientos, especificaciones, documentación del usuario y muchos otros elementos del costo tienden a comportarse como costos fijos y por lo tanto, provocan que métricas tales como "costo por líneas de código", paradójicamente, aumente en lugar de disminuir [Jones'97].

Albrecht estableció que la unidad económica resultante de los proyectos de software debía ser válida para todos los lenguajes y representar tópicos concernientes a los usuarios del software. Brevemente, deseaba medir la funcionalidad del software. Albrecht consideraba que los aspectos externos visibles del software que se podrían enumerar con precisión consistían en cinco ítems: las entradas hacia la aplicación, las salidas desde ella, las consultas de los usuarios, los archivos de datos que pudieran actualizarse y las interfases hacia otras aplicaciones. A partir de un proceso de prueba y error, surgieron los factores de peso empíricos para los cinco ítems, que representan la dificultad de implementar cada uno de ellos. Esto marcó un hito en la historia de la computación: que se pudiera medir la productividad económica del software [Jones'97].

El FPA es una medida de clara significación para los sistemas administrativos. La técnica FPA cuantifica las funciones contenidas dentro del software en términos que son significativos para los usuarios del software. Esta medida se relaciona directamente a los requerimientos de ese tipo de aplicaciones. Se puede aplicar a un amplio rango de entornos de desarrollo y a través del ciclo de vida de un proyecto de desarrollo, desde la primera definición de los requerimientos hasta el uso operacional completo, permitiendo refinar la estimación de tamaño y por lo tanto, las estimaciones de costo o productividad [IFPUG'00[b]] [Fenton'96].

Los FP miden el tamaño del software cuantificando la funcionalidad provista al usuario basándose solamente en el diseño lógico y las especificaciones funcionales [IFPUG'00[a]].

El cálculo de FP se puede realizar con documentación mínima. Sin embargo, la precisión y eficiencia de las cuentas mejoran con documentación apropiada. Ejemplos de documentación apropiada son [Heller'95]:

- Especificación de diseño.
- Visualización del diseño.
- Requerimientos de datos (internos y externos).
- Descripción de las interfases del usuario.

En general se tiende a pensar en FPA como una técnica para realizar mediciones de tamaño que permite realizar comparaciones de productividad a través de proyectos, sistemas y empresas o para estimaciones de costos, sin embargo, el factor más importante es que FPA visualiza a los sistemas desde la perspectiva del usuario. Los usuarios ingresan información a la aplicación, obtienen información y reconocen grupos discretos de datos, entidades o archivos que el sistema usa y mantiene. Al descomponer el sistema en sus partes, se puede alcanzar el más bajo nivel de funcionalidad reconocible por el usuario (Figura 13) [Goodman'99].

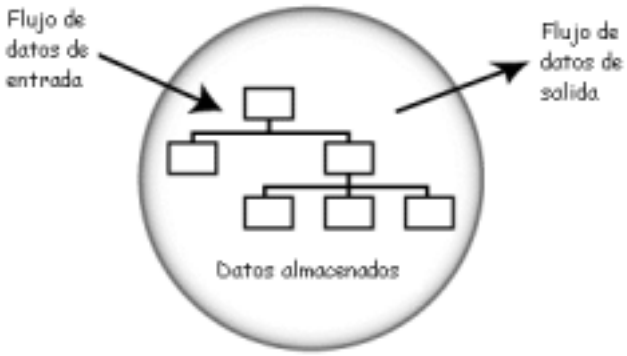

Figura 13 – Visión del sistema desde la perspectiva del usuario [Goodman'99]

La medición de los requerimientos ayuda a los ingenieros a producir requerimientos no ambiguos y medibles, además de mejorar el rigor con que éstos son documentados. En consecuencia, pueden usarse como base para los acuerdos entre clientes y proveedores [Grant Rule'01[b]].

Con respecto a esto último, es oportuno destacar los siguientes párrafos [Grant Rule'01ª]:

- Los usuarios pagan para que sean satisfechos sus requerimientos; el código no es su preocupación, para un usuario el producto entregable es la satisfacción de sus requerimientos.
- Las medidas del *tamaño funcional* expresan la cantidad de procesamiento de información entregada: la comunidad del software necesita una medida de la cantidad de funcionalidad que el cliente requiere del software, independientemente de la tecnología usada y de la gente que lo produce. Esas medidas son de importancia crucial, no sólo para la administración del proyecto, sino para tomar decisiones económicas.
- El tamaño es relevante desde la definición de requerimientos hasta la entrega final: los proveedores de software usualmente tienen presiones para estimar el esfuerzo y los costos de desarrollo al principio del ciclo de vida de un proyecto. A menudo hay necesidad de establecer compromisos basados en una inadecuada comprensión de los requerimientos del cliente. Cualquier técnica que ayude a remover la ambigüedad, mejorar la definición de los requerimientos y que permita a clientes y proveedores acordar los términos y controlar el alcance y progreso de los contratos, será bienvenido por la comunidad del software.

Se han desarrollado varios métodos de FPA, entre ellos el de Albrecht, MarkII, COSMIC y unas cuantas variaciones propuestas por otros autores. En la Figura 14 se muestra la evolución de los principales métodos.

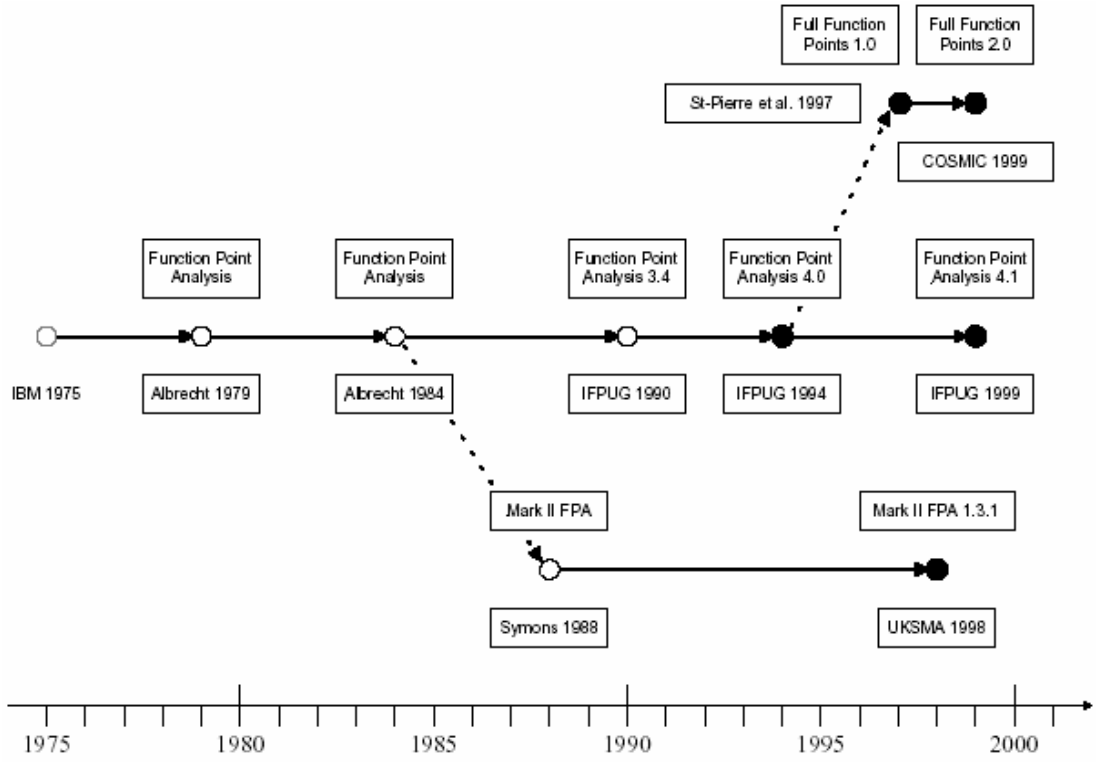

Figura 14 - La evolución de los métodos de medición del tamaño funcional [Fetcke'99]

Debido a la amplia difusión de las métricas de FP surgió la necesidad de definir estándares. En 1996 la International Standards Organisation inició un grupo de trabajo (ISO/IEC JTC1 SC7 WG12) con la participación de expertos en métricas del software para establecer los principios comunes de la medición del *tamaño funcional*. La primera publicación que estableció los principios básicos fue ISO/IEC 14143-1 [Symons'01]. Los métodos IFPUG, MKII FPA y NESMA (variante del método IFPUG mantenido por Netherlands Software Metrics Association) han sido aprobados por ISO. En febrero de 2003 se aprobó el estándar ISO/IEC 19761:2003 para el método COSMIC –FFP [COSMIC-FFP'03].

Se describirán en detalle los métodos de Albrecht por tratarse del precursor de la idea y MarkII que será la base para el desarrollo del enfoque propuesto en esta tesis.

La Tabla 1 muestra un cuadro comparativo con algunos aspectos de los métodos.

Consideración	SLOC	Albrecht FP	MarkII FP
Definición			
¿Estándar aceptado?	No, varios	Sí, se está refinando	Sí, recomendado por CCTA para UK Gov.
¿Claridad de la definición?	Potencialmente preciso	Subjetivo en parte	Razonablemente objetivo
¿Basado en métodos estructurados?	No	No	Sí
Usabilidad			
Facilidad de uso	Fácil	← No tan fácil como parece →	
¿Automatizado?	Sí	Muy difícil	Sí (herramienta CASE)
¿Usable para estimación?	En algunas circunstancias	Sí	Sí
Disponibilidad			
Público/propietario	Ambas	Público	Público desde 1991
Base de usuarios	Muy amplio	Muy amplio	Adecuado
Grupo de usuarios	Varios (Ej. COCOMO User Groups)	Varios (Ej. IFPUG)	Sí (Ej. EFPUG)
Entrenamiento público	No disponible	Sí	Sí

Tabla 1 - Comparación entre los métodos [Symons'91]

3.7.2. Proceso de medición del tamaño funcional [Fetcke'99]

La medición puede entenderse como una abstracción que captura ciertos atributos de los objetos de interés. La teoría de la medición visualiza esta abstracción como un

mapeo que asigna objetos numéricos a objetos empíricos. En la medición de *tamaño funcional*, esos objetos empíricos son las aplicaciones de software. Esta técnica de medición requiere dos pasos de abstracción.

Los dos pasos de abstracción

En lugar de usar los conceptos y modelos de un método de desarrollo particular, cada variante del FPA define sus propios conceptos para la representación de una aplicación de software. Esos métodos, por lo tanto, definen una abstracción del software que representa los ítems considerados relevantes para el *tamaño funcional*. Las variantes de FPA que se muestran en la Figura 14 se basan en una abstracción orientada a los datos. Esos métodos definen los dos pasos de abstracción siguientes:

1. Los requerimientos funcionales del usuario están representados en la abstracción orientada a los datos.
2. Los ítems de la representación orientada a los datos se asocian con números.

El primer paso de la abstracción se aplica a la documentación de la funcionalidad del software. El resultado es una representación que contiene los ítems significativos para el *tamaño funcional*. Este paso requiere de la evaluación de reglas por parte del profesional.

El segundo paso es la medición, es decir, el mapeo de esos ítems con números. La fuente para este paso es la abstracción orientada a los datos.

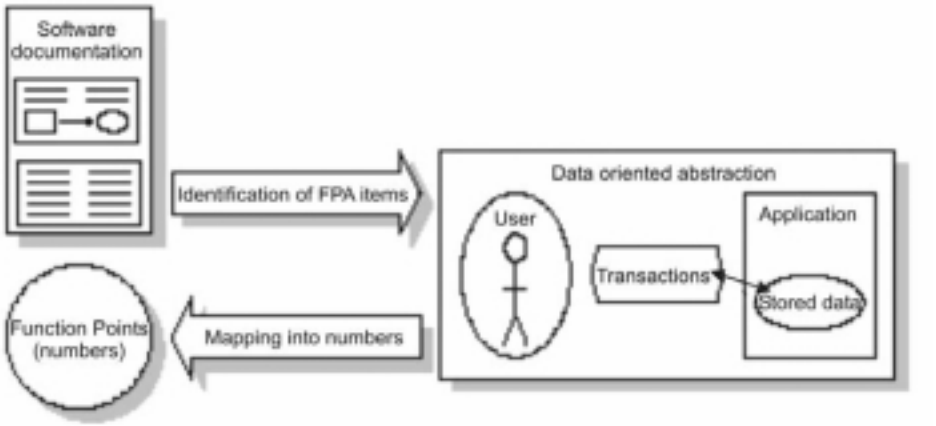

Figura 15 - Los dos pasos de abstracción de las variantes del FPA [Fetcke'99]

La Figura 15 ilustra la visión de los dos pasos de la abstracción orientada a los datos. Ambos pasos están definidos por las reglas de los métodos de medición del *tamaño funcional*. Sin embargo, en la documentación de los mismos no suele aparecer tan claramente la separación entre esos dos pasos de la abstracción.

La abstracción orientada a los datos

El IFPUG FPA plantea una abstracción orientada a los datos de la aplicación medida. Se han propuesto variantes del FPA para mejorar el enfoque original. Estas variantes difieren en los ítems identificados y en las funciones medidas.

Particularmente el enfoque MarkII se basa en los mismos conceptos centrales que el estándar IFPUG [Fetcke'99].

- *Concepto de usuario*. El usuario interactúa con una aplicación y no necesariamente se restringe a usuarios humanos, sino que puede incluir aplicaciones de software y hardware.
- *Concepto de aplicación*. La aplicación es el objeto de la medición. Las aplicaciones proveen funciones al usuario. Esas funciones son los atributos de interés, precisamente, los requerimientos que especifica el usuario para esas funciones.
- *Concepto de transacción*. Las *transacciones* son procesos de interacción del usuario con la aplicación desde una perspectiva lógica o funcional. En términos de IFPUG FPA, las *transacciones*:
  - son las unidades de actividad más pequeñas significativas para el usuario.
  - son autocontenidas, es decir lógicamente completas y
  - dejan la aplicación en estado consistente.
- *Concepto de dato*. Los datos son almacenados por la aplicación. Los elementos dato representan los ítems de datos más pequeños significativos para el usuario y están estructurados en grupos lógicamente relacionados, similares a las tablas en una base de datos.
- *Concepto de tipo*. En el nivel más alto de la abstracción orientada a los datos, todas las variantes de FPA identifican tipos de *transacciones* y tipos de grupos de datos. El término *tipo* se refiere al principio que múltiples instancias del mismo componente lógico se identifican sólo una vez. Este concepto es esencial para la visión del *tamaño funcional* en las variantes de FPA.

  La Figura 16 ilustra estos conceptos de la abstracción orientada a los datos.

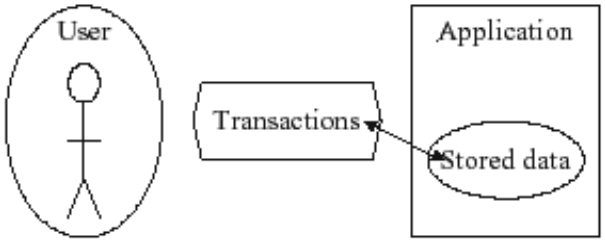

Figura 16 – La abstracción orientada a los datos del FPA [Fetcke'99]

3.7.3. FPA y los requerimientos

Hay una diversidad de enfoques tradicionales para identificar y obtener los requerimientos del software que usados adecuadamente permiten entregar una forma documentada de los requerimientos del usuario. Después de una serie de revisiones con el usuario, se asume que esos requerimientos formales representan el conjunto completo de requerimientos del usuario. Sin embargo, el conjunto completo de requerimientos del usuario generalmente no está completo hasta el final del proyecto y continúa evolucionando mientras éste progresa. La experiencia indica que FPA es

efectivo para identificar el conjunto completo de requerimientos funcionales del usuario y descubrir defectos potenciales. De hecho, los beneficios obtenidos por la aplicación de FPA a los requerimientos funcionales del usuario pueden ser más valiosos que el mero tamaño en FP del software [Dekkers'01].

En este punto vale la pena identificar los principales tipos de requerimientos del software y el aporte de FPA en cada caso. Primero están los requerimientos funcionales del usuario, que son las funciones que el sistema debe ejecutar. Todos los sistemas desde los de tiempo real hasta los sistemas administrativos tienen requerimientos funcionales. Estos son los procesos elementales que deben realizarse para la entrada, proceso, manipulación, salida e interfases de datos desde y hacia el software. FPA soporta este tipo de requerimientos del usuario. Los segundos son los requerimientos no-funcionales. Éstos son las restricciones independientes de la tecnología y de la empresa, que el software debe considerar. Estos requerimientos incluyen la calidad y los requerimientos de performance como portabilidad, usabilidad, seguridad, confiabilidad y velocidad. Parte de las técnicas de FPA pueden asistir con ese tipo de requerimientos. Por ultimo, los requerimientos técnicos que son los requerimientos del usuario para una determinada configuración de hardware/software o una configuración técnica que debe ser entregada. Por ejemplo, los requerimientos técnicos pueden especificar una base de datos determinada o una solución de hardware concurrente. FPA no soporta este tipo de requerimientos [Dekkers'01].

3.7.4. Beneficios de la aplicación del FPA [Goldfarb'00]

El FPA provee las bases para:

Mejorar la definición de los requerimientos. Las técnicas FPA ayudan a los ingenieros a producir requerimientos del software que son mensurables. Reducen la ambigüedad y mejoran el rigor con que son documentados los requerimientos. Se pueden usar como base para el acuerdo entre clientes y proveedores [Grant Rule'01ª].

Comunicar requerimientos funcionales. Este fue el objetivo original subyacente del desarrollo de los FP. A partir de evitar la terminología técnica y enfocar sobre los requerimientos del usuario, resulta un excelente vehículo para comunicarse con los usuarios. La técnica puede usarse para realizar entrevistas a los clientes y documentar los resultados obtenidos. La documentación resultante provee un marco (framework) que describe los requerimientos del usuario y técnicos [Heller'95].

Estimar el esfuerzo, agenda y costos basándose en los requerimientos. Cuando un cliente requiere una nueva aplicación de software desarrollada por un proveedor, necesita una estimación del costo de desarrollo a medida que evolucionan los requerimientos, para asegurar la decisión acerca de la óptima inversión costo beneficio. Para determinar el precio el proveedor debe estimar el esfuerzo de desarrollo, el personal necesario y por lo tanto, el costo resultante, comenzando solamente desde el tamaño de los requerimientos. Esas estimaciones se necesitan al principio, pero claramente cuanto más temprano se hacen son más inciertas. En consecuencia, la estimación necesita repetirse, mejorando la precisión cuando la

comprensión de los requerimientos es más detallada [Grant Rule'01ª]. Dentro de los varios factores que necesitan considerarse para estimar un proyecto de software, los dos puntos claves esenciales son: el tamaño del producto entregable y cuánto de ese producto se puede producir en un período definido de tiempo. El tamaño se puede derivar desde los FP y el segundo requerimiento para la estimación se determina a partir del tiempo que toma producir un FP [Heller'95].

Evaluar y administrar la factibilidad de un proyecto. En una organización cuyos recursos están limitados en términos de fondos, habilidades, disponibilidad, tiempo u otro recurso, lo que pueda lograrse estará determinado por la cantidad de recursos y la capacidad de la organización para producir software. La factibilidad de un proyecto estará restringida por la capacidad en términos de cantidad de resultados que se pueden producir para un cierto desembolso en un tiempo dado. La experiencia adquirida con el FPA en proyectos previos permite hacer estimaciones antes de considerar en detalle los requerimientos de uno nuevo [Grant Rule'01ª].

Administrar los cambios. Este es otro beneficio clave del FPA para un proyecto. Una vez que el proyecto ha sido aprobado y se han calculado los FP, es una tarea relativamente fácil identificar, rastrear y comunicar los requerimientos nuevos o que han sido cambiados. Cuando el usuario propone un cambio en los requerimientos, se desarrollan los FP y el resultado se usa para determinar el impacto en el esfuerzo y presupuesto del proyecto. El usuario y el equipo de proyecto pueden determinar la importancia del cambio requerido frente al impacto en el presupuesto y la agenda. A la finalización del proyecto se puede evaluar la cuenta final de FP frente a la estimación inicial para determinar la efectividad de las técnicas para obtener requerimientos. Este análisis ayuda a identificar oportunidades para mejorar el proceso de definición de requerimientos [Heller'95].

Mejorar el mantenimiento y soporte de la aplicación. Los FP se pueden usar para evaluar el soporte a los requerimientos de mantenimiento de sistemas. En este análisis la productividad se determina calculando la cantidad de FP que un individuo puede mantener para un sistema dado en el término de un año. Cuando se compara con otros sistemas, esa tasa ayuda a identificar qué sistemas requieren mayor soporte. Ese análisis ayuda a la organización a desarrollar una estrategia de mantenimiento y reemplazo para esos sistemas [Heller'95].

Medir la productividad. Es una salida natural del FPA. Puesto que los FP son independientes de la tecnología, se pueden usar para comparar la productividad a través de diferentes herramientas y plataformas. Más importante aún, pueden usarse para establecer una tasa de productividad para un conjunto de herramienta y plataforma específica. Una vez que esa tasa de productividad está establecida, puede usarse para estimar proyectos y rastrear a lo largo del tiempo para determinar el impacto que tienen las iniciativas de mejoramiento del proceso sobre la productividad [Heller'95].

Verificar la completitud de los requerimientos. Para verificar la completitud, uno de los problemas es asegurar que se ha identificado el conjunto completo de requerimientos. Para ello, los analistas toman como marco de referencia los modelos

teóricos del análisis estructurado, modelado de datos y/o la experiencia adquirida a partir de otros sistemas similares. FPA provee un marco de referencia adicional para verificar la completitud de los requerimientos funcionales basado en una perspectiva enfocada al usuario. FPA examina el conjunto de requerimientos en términos de datos y movimiento/manipulación de los mismos *(transacciones)*, tal como son entendidos y expresados por los usuarios y sobre esta base determina el *tamaño funcional* del software. Cada uno de los pasos del FPA ayuda a clarificar los requerimientos, permitiendo confirmar los requerimientos originales o revelando las inconsistencias en la comprensión. El aporte a la completitud va mucho más allá de la fase de requerimientos. Simplemente revisando el listado de FP se puede determinar si una cierta funcionalidad está incluida y ayudar a tomar decisiones acerca de incluirla o no [Dekkers'01].

3.7.5. Método de Albrecht

El primer método basado en métricas de funcionalidad fue propuesto por Alan Albrecht. La propuesta partía de un concepto distinto, el desarrollo de un índice de la funcionalidad de un sistema como una medida asociada a su tamaño [Symons'91]. Desde entonces, ha aumentado el uso del FPA en el mundo del desarrollo de sistemas y los métodos se han perfeccionado basándose en la experiencia acumulada. Existe una organización, el International Function Point Users Group (IFPUG) que se ocupa de mantener, actualizar y difundir la información y la práctica.

La idea es que se pueden calcular FP sin forzar a la especificación a ajustarse a un modelo o técnica de especificación [Fenton'96]. El proceso de medición se realiza en el contexto de un modelo abstracto del sistema. Ese modelo se compone de *transacciones* y *archivos*. Estos elementos se identifican a partir de los documentos de requerimientos, técnicas de diseño estructurado u otros modelos.

Cuando Albrecht desarrolló el FPA tenía por objetivos lograr [Symons'91]:

- Una medida consistente del tamaño del sistema para usar en mediciones o estimaciones de productividad.
- Un método significativo para el usuario final. Al estar relacionados directamente a los requerimientos del usuario, los FP deberían ser más fáciles de comprender que las SLOC.
- Reglas para contar los FP que deberían ser fáciles de aplicar.
- FP que deberían ser estimables a partir de la especificación de requerimientos.
- FP que deberían ser independientes de la tecnología usada para desarrollar el sistema.

Los seres humanos resuelven los problemas descomponiéndolos en piezas más pequeñas que les resultan más comprensibles. Los problemas que pueden parecer difíciles, son simples una vez que son divididos en partes. Clasificar las cosas, colocarlas en alguna categoría, es un proceso corriente. Cuando los objetos a clasificar son los contenidos de sistemas, se debe usar un conjunto de definiciones y

reglas para colocar a esos objetos en la categoría apropiada. El FPA es una técnica estructurada de clasificación de los componentes de un sistema [Longstreet'96ª].

En el mundo del FPA, los dos componentes del tamaño del sistema de la Figura 11, están representados por cinco grandes clases el primero y por las denominadas características generales del sistema el segundo [Longstreet'96ª].

El componente tamaño del procesamiento se descompone en: *entradas externas (External Inputs)*, *salidas externas (External Outputs)* y *consultas externas (External Inquires)*, cada una de las cuales interactúa con archivos y son llamadas *transacciones*. Los otros dos son los *archivos lógicos internos (Internal Logical Files)* y los *archivos de interfase externa* hacia otras aplicaciones *(External Interface Files)* [Symons'91].

El componente requerimientos técnicos está representado por las características generales del sistema que valoran la funcionalidad general del sistema [Symons'91].

En la Figura 17 se visualizan gráficamente esos componentes.

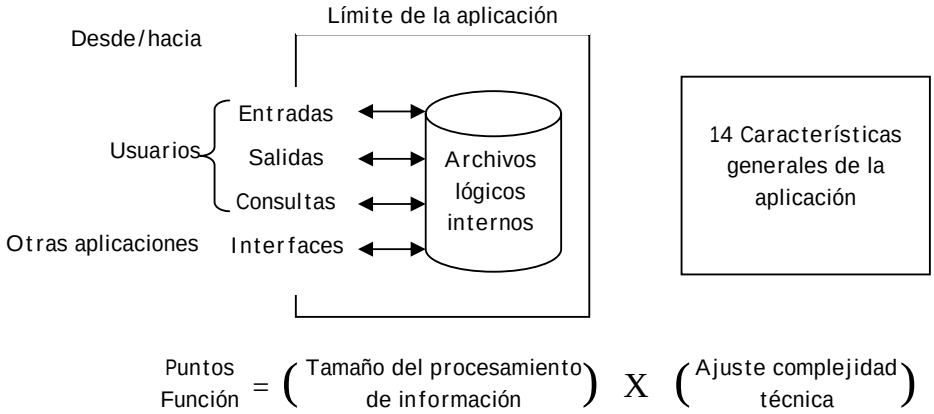

Figura 17 - Componentes del Método de Puntos Función de Albrecht [Symons'91]

Debido a que los FP miden los sistemas desde una perspectiva funcional, son independientes de la tecnología. Independientemente del lenguaje, método de desarrollo o plataforma de hardware usados, la cantidad de FP permanecerá constante. La única variable es la cantidad de esfuerzo necesario para entregar un conjunto de FP. Por lo tanto, el FPA se puede usar para determinar si una herramienta, un entorno o un lenguaje es más productivo comparado con otros, dentro de una organización o entre organizaciones [Longstreet'96ª].

El componente tamaño del procesamiento: los cinco principales componentes

Debido a que es común que los sistemas interactúen con otros sistemas, debe trazarse un *límite* alrededor del sistema a medir, previo a clasificar sus componentes. Este *límite* se debe trazar de acuerdo al punto de vista del usuario e indica el borde entre el proyecto o aplicación que se está midiendo y las aplicaciones externas o el dominio del usuario [Longstreet'96ª].

Entradas externas (*EI*) es un proceso elemental en el cual los datos cruzan el *límite* desde el exterior hacia el interior del sistema. Estos datos pueden venir desde una pantalla de entrada de datos o desde otra aplicación. Los datos pueden ser usados para mantener uno o más *archivos lógicos internos*. Ver Figura 18 [Longstreet'96[a]] [Longstreet'96[b]].

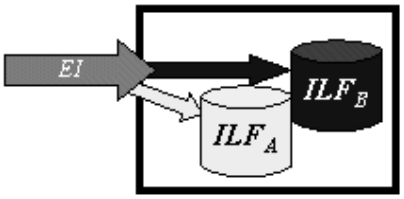

Figura 18 - EI que actualiza 2 ILF

Salidas externas (*EO*) es un proceso elemental en el cual los datos derivados atraviesan el *límite* desde el interior hacia el exterior del sistema. Además, un *EO* puede actualizar un *archivo lógico interno*. Los datos crean reportes o archivos de salida hacia otras aplicaciones. Esos reportes y archivos son creados a partir de uno o más *archivos lógicos internos* y *archivos de interfase externa*. Ver Figura 19 [Longstreet'96[a]] [Longstreet'96[b]].

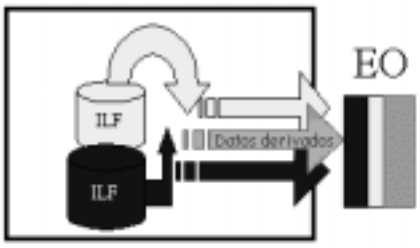

Figura 19 - EO que referencia y deriva información de 2 ILF

Consultas externas (*EQ*) es un proceso elemental con componentes de entrada y salida que resulta en el retorno de datos desde uno o más archivos lógicos internos y archivos de interfases externas. El proceso de entrada no actualiza archivos *ILF* y la salida no contiene datos derivados. Ver Figura 20 [Longstreet'96[b]].

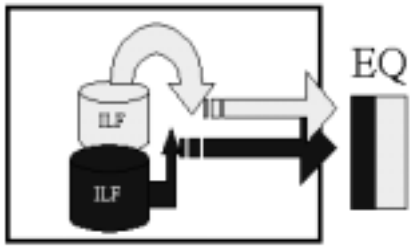

Figura 20 - EQ con 2 ILF y sin datos derivados

Archivos lógicos internos (*ILF*) es un grupo de datos lógicamente relacionados, identificable por el usuario, que residen completamente dentro del *límite de la aplicación* y es mantenido a través de entradas externas [Longstreet'96[a]] [Longstreet'96[b]].

Archivos de interfase externa (*EIF*) es un grupo de datos lógicamente relacionados, identificable por el usuario, que se usa solamente con propósitos de *referencia*. Los datos residen completamente fuera de la aplicación y son mantenidos por otra aplicación [Longstreet'96[a]] [Longstreet'96[b]].

Una vez que todos los componentes han sido clasificados como *EI*, *EO*, *EQ*, *ILF* o *EIF*, se les asigna una posición en la escala que puede ser: *bajo*, *medio* o *alto (Low, Average o High)*. Para las *transacciones* (*EI, EO, EQ*) esa posición se basa en la cantidad de *archivos actualizados o referenciados* (*FTR*) y la cantidad de *tipos de elemento dato (Data element type - DET)*. Un *DET* es un campo único, no recursivo, reconocible por el usuario. En el caso particular de las *EQ* se ubican en la escala *(bajo, medio o alto)* como *EO*, pero se les asigna un factor como *EI*. Si el mismo *FTR* se usa en la parte de entrada y salida se cuenta una sola vez. Si el mismo *DET* se usa en la parte de entrada y salida, también se cuenta sólo una vez [Longstreet'96[a]].

Cada una de las siguientes tablas se utiliza en el proceso de asignación de posiciones en la escala y sus factores asociados:

FTR	Elementos dato		
	1-4	5-15	>15
0-1	Bajo	Bajo	Medio
2	Bajo	Medio	Alto
3 o más	Medio	Alto	Alto

Tabla 2 - EI [Longstreet'96[a]]

FTR	Elementos dato		
	1-5	6-19	>19
0-1	Bajo	Bajo	Medio
2-3	Bajo	Medio	Alto
> 3	Medio	Alto	Alto

Tabla 3 - EO y EQ [Longstreet'96[a]]

Posición	Factor		
	EO	EQ	EI
Bajo	4	3	3
Medio	5	4	4
Alto	7	6	6

Tabla 4 - Transacciones [Longstreet'96[a]]

Para los archivos *ILF* y *EIF* la posición en la escala *(bajo, medio, alto)* se basa en los *tipos de elementos registro* (*RET*) y *DET*. Un *RET* es un subgrupo de datos reconocible por el usuario dentro de un *ILF* o *EIF* [Longstreet'96[a]].

RET	Elementos dato		
	1-19	20-50	>50
1	Bajo	Bajo	Medio
2-5	Bajo	Medio	Alto
>5	Medio	Alto	Alto

Tabla 5 – Tipo registro [Longstreet'96[a]]

Posición	Factor	
	ILF	EIF
Bajo	7	5
Medio	10	7
Alto	15	10

Tabla 6 – ILF y EIF [Longstreet'96[a]]

Las cuentas para cada nivel de complejidad de cada tipo de componente se pueden ingresar en una tabla como la siguiente (Tabla 7). Cada cuenta se multiplica por el puntaje numérico indicado para determinar el puntaje total. Los valores de cada fila se suman a través de la tabla resultando el valor total para cada tipo de componente. Esos totales luego se suman a través de la tabla para obtener la cantidad total de *FP sin ajustar* (*Unadjusted Function Points - UFP*) [Longstreet'96[a]].

Tipo de componente	Complejidad de los componentes			
	Bajo	Medio	Alto	Total
Entradas Externas (EI)	x 3 =	x 4 =	x 6 =	
Salidas Externas (EO)	x 4 =	x 5 =	x 7 =	
Consultas Externas (EQ)	x 3 =	x 4 =	x 6 =	
Archivos lógicos internos (ILF)	x 7 =	x 10 =	x 15 =	
Archivos de interfase externa (EIF)	x 5 =	x 7 =	x 10 =	
		UFP		
		TCA		
		FP		

Tabla 7 – Cálculo de Puntos Función de Albrecht [Longstreet'96[a]]

El componente complejidad técnica

El valor de *Ajuste de complejidad técnica (TCA)* se obtiene a partir de las 14 características generales del sistema que asignan un puntaje a la funcionalidad general de la aplicación que se está midiendo. Esas características tienen descripciones asociadas que ayudan a determinar los *grados de influencia (DI)* de las características. Los *grados de influencia* varían en una escala de cero a cinco, desde ninguna a fuerte influencia. El objetivo de este factor es considerar la influencia sobre el *tamaño funcional* de los requerimientos de calidad y técnicos. En la Tabla 8 se detallan las características generales [Longstreet'96ª].

Características Generales del Sistema			
1	Comunicación de datos	8	Actualización interactiva
2	Procesamiento de datos distribuido	9	Complejidad de procesamiento
3	Performance	10	Reusabilidad
4	Entorno operativo muy utilizado	11	Facilidad de instalación
5	Frecuencia de transacción	12	Facilidad de operación
6	Entrada de datos interactiva	13	Múltiples instalaciones
7	Eficiencia usuario final	14	Facilidad de cambios

Tabla 8 - Características Generales del Sistema para Albrecht FPA [Longstreet'96ª]

Una vez que han sido valorados los 14 factores, debe aplicarse la ecuación de *Ajuste de complejidad técnica (TCA)* del IFPUG.

$$TCA = 0.65 + 0.01 * \sum DI$$

donde *DI* son los *grados de influencia* dados por cada una de las 14 características generales del sistema.

La cuenta final de FP se obtiene multiplicando el valor UFP por TCA.

$$FP = UFP * TCA$$

Se completa la Tabla 7 con el valor TCA y se obtiene FP.

Algunas críticas al Método de Albrecht

A pesar de ser una idea muy interesante y novedosa, al experimentar con el uso del método de Albrecht surgieron una cantidad de dificultades y anomalías. Entre ellas se puede mencionar [Symons'91]:

- Los componentes propuestos por Albrecht a veces son difíciles de identificar en la práctica del desarrollo de sistemas.
- Es razonable preguntarse cómo determinó Albrecht los pesos (o puntajes) usados para los diferentes tipos de componentes y para los diferentes niveles de complejidad cuando se calculan los *UFP*.

- Una crítica similar se puede hacer respecto a la elección de los componentes y los pesos para las características generales de la aplicación del *Ajuste de complejidad técnica*.
- Otra cuestión que se plantea es si este método trata adecuadamente con la complejidad de procesamiento interno del sistema.
- Por último, las reglas de Albrecht parecen favorecer a los sistemas discretos frente a los sistemas integrados.

Las más serias deficiencias del Método de Albrecht son aquéllas que surgen a partir del estrecho rango de puntajes en FP que se puede adjudicar a *transacciones* con un rango muy amplio de variación en la cantidad de *DET* en la entrada/salida y en la complejidad de procesamiento interno [Symons'91].

A partir de este método y de las críticas que se han expuesto, Charles Symons desarrolló un nuevo enfoque basándose en el mismo concepto. Este método se conoce como Análisis de Puntos Función MarkII o simplemente MKII FPA.

3.7.6. Método MarkII

Este método fue desarrollado por Symons a fines de los '80 y tiene la misma finalidad y estructura que el de Albrecht, pero supera las dificultades antes enunciadas. La organización internacional que se ocupa de mantener, actualizar y difundir la información es United Kingdom Software Metrics Association (UKSMA). El método MKII cumple con la norma ISO 14143/1 [MKII FPA CPM'98].

"Simple in concept, easy to apply, aligned with modern systems analysis methods, for intelligent software sizing and estimating" [MKII FPA CPM'98]

Este método puede usarse para medir el *tamaño funcional* de cualquier aplicación de software que se pueda describir en términos de *transacciones lógicas*, cada una comprendiendo componentes de entrada, proceso y salida. Las reglas fueron diseñadas para aplicarlas a sistemas de información de gestión, donde el componente procesamiento de cada *transacción* tiende a estar dominado por consideraciones de almacenamiento y recuperación de datos. Puede ser aplicable a otros dominios de software, pero debe notarse que las reglas no toman en cuenta las contribuciones al tamaño de algoritmos complejos como aquéllos que se encuentran en el software científico y de ingeniería, ni toman en cuenta los requerimientos de tiempo real [MKII FPA CPM'98].

En la Figura 21 se describen los componentes del Método MKII. Los cambios más significativos respecto al método descripto anteriormente están relacionados al tamaño del componente procesamiento de información. La idea es considerar al sistema como una colección de *transacciones lógicas* [Symons'91]. Para ello es necesario determinar previamente el *límite del sistema*, a fin de establecer cuales pertenecen al sistema que se medirá y cuales quedarán excluidas. La aplicación o parte de la aplicación incluida dentro del *límite* debe ser un cuerpo coherente de funcionalidad que comprende una o más *transacciones lógicas* [MKII FPA CPM'98].

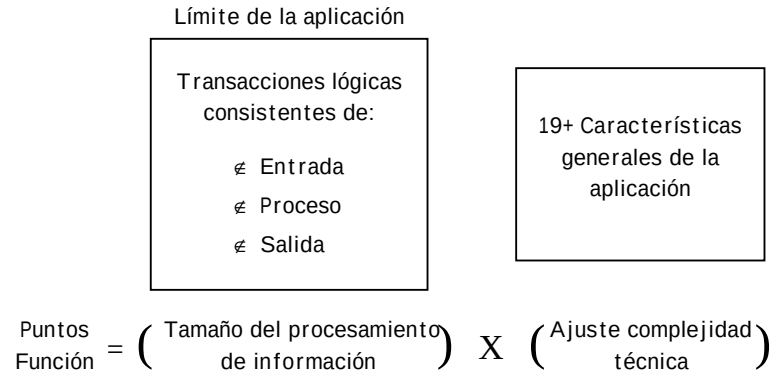

Figura 21 - Componentes del Método de Puntos Función MarkII [Symons'91]

Una *transacción lógica* se define como una combinación de tres componentes [MKII FPA CPM'98]:

- ∉ Un elemento de entrada a través del *límite*.
- ∉ Un elemento de procesamiento de los datos dentro del *límite*.
- ∉ Un elemento de salida a través del *límite*.

Tamaño funcional y transacciones lógicas

El *tamaño funcional* de una aplicación es la suma de los tamaños de cada *transacción lógica*, cuyos componentes de entrada y salida atraviesan el *límite de la aplicación*.

La *transacción lógica* es el proceso autoconsistente de más bajo nivel. Corresponde a un proceso lógico único e independiente. Es disparada por un único *evento* del mundo externo significativo para el usuario, que al completarse deja la aplicación en estado consistente con relación al *evento*. También puede ser disparada por un requerimiento del usuario para extraer información desde la aplicación sin modificar los datos almacenados [MKII FPA CPM'98]. Las *transacciones* son procesos completos. Una *transacción lógica* se cuenta sólo una vez, aún aunque pueda ser ejecutada en más de un punto en la aplicación.

Componente procesamiento en las transacciones lógicas

El tamaño de este componente es proporcional a la cantidad de entidades de datos *referenciadas* en cada *transacción*, entendiéndose como *referencia* a los procesos de creación, lectura, actualización, eliminación, listado. Se cuenta la cantidad de *tipos de entidades primarias* que son *referenciadas* por la *transacción lógica*, más la *referencia* a una entidad denominada *System* (que se describirá más adelante), si es necesario [MKII FPA CPM'98].

El concepto de *entidad* se define como: alguna cosa en el mundo real (objeto, transacción, período de tiempo, tangible o intangible) acerca de la cual se requiere mantener información. Una *entidad* es un elemento cuyas ocurrencias son identificables en forma individual o colectiva [Symons'91].

Todos los accesos a *tipos de entidades no primarias* se consideran *referencias* a una única entidad llamada *System*. Dentro del *límite de la aplicación* puede haber solo una entidad *System* y como máximo se puede incluir una *referencia* en el componente procesamiento de una *transacción lógica* [MKII FPA CPM'98].

Distinguir los tipos de entidad primaria y no primaria: la entidad System

Las *entidades primarias* son aquéllas que representan elementos principales para el sistema en estudio. Las *entidades no primarias* son aquéllas que tienen pocos atributos y generalmente tienen relación con tablas maestras que sirven para propósitos de validación. Se agrupan en una única entidad *System*. La distinción entre ambas clases de *entidades* no es absoluta, puede haber casos en el límite entre ambas [Symons'91].

Hay varios criterios para distinguir *entidades primarias* y *no primarias* [MKII FPA CPM'98] [Symons'91]:

Criterio	Entidad primaria	Entidad no primaria
Número de atributos	Varios, con valores que cambian frecuentemente	Muy pocos; los valores cambian raramente
Frecuencia de ocurrencia	Puede cambiar a menudo	Fijo permanentemente o cambia raramente
Tiempo de cambio de los valores atributos	Durante la operación normal del sistema	Usualmente fuera de la operación normal del sistema
Autorización de cambios de los valores atributos	Ninguna especial, ejecutados por el usuario normal del sistema	Ejecutados por el administrador del sistema
Ciclo de vida de la entidad	Usualmente varias etapas o estados	Usualmente sólo "vive" o no existente

Tabla 9 - Criterios para distinguir entidades primarias [Symons'91]

Componentes de entrada y salida en las transacciones lógicas

Este método asume que el tamaño relativo de los componentes entrada y salida en cada *transacción lógica* es proporcional a la cantidad de *DET* en cada componente respectivo. Un *DET* es un ítem único de información que es indivisible para el propósito de la *transacción lógica* y es parte de un flujo de datos en la *entrada* y *salida* a través del *límite* [MKII FPA CPM'98].

Pasos para calcular el Índice de Puntos Función

Para evaluar el *Índice de Puntos Función (FPI)*, se deben seguir los siguientes pasos [Symons'91] [MKII FPA CPM'98]:

1. *Obtener una comprensión general del sistema al cual se aplicará el método*. Para lograr el conocimiento del problema y del sistema necesario para cumplir con los requerimientos del usuario, se realiza la Especificación de Requerimientos.

2. *Definir el punto de vista, propósito de la medición y límite del sistema* que se medirá, para establecer qué elementos estarán incluidos dentro del mismo.
3. *Determinar las entidades primarias y no primarias.* Es necesario distinguir las *entidades primarias* y *no primarias* agrupadas en la entidad *System*.
4. *Determinar las transacciones lógicas*.

   Siempre hay que tener presente: *¿Qué requiere el usuario?*

   Hay algunas reglas que permiten diferenciar algunos procesos para determinar si se consideran o no *transacciones* [Symons'91].

   - Los procesos internos que se repiten con frecuencia deben ser contados en cada *transacción* que ocurren. No deben contarse como *transacciones lógicas* separadas, a menos que sean disparados en forma independiente por un *evento* significativo para el usuario.
   - Los cambios en el valor de un *DET* de entrada no pueden disparar *transacciones lógicas* separadas.
   - Los *eventos* que son significativos para el diseñador físico del sistema no se cuentan como *eventos* que disparan *transacciones lógicas*.

5. *Determinar los componentes de las transacciones lógicas*. Cada una de ellas consta de tres componentes: entrada, procesamiento y salida [MKII FPA CPM'98]:

   5.1. El componente entrada a través del *límite* consiste de la adquisición y validación de los datos ingresados que describen un *evento* de interés en el mundo externo o de los parámetros de una *consulta* para obtener información de salida desde la aplicación.

   5.2. El componente procesamiento consiste del almacenamiento y recuperación de información que describe el estado de las entidades de interés en el mundo externo.

   5.3. El componente salida a través del *límite* consiste del formateo y presentación de información hacia el mundo externo.

   Para organizar el trabajo se utiliza una tabla de *transacciones lógicas* con el siguiente encabezado:

ID	Nombre transacción	Evento o consulta	Nº de DET entrada	Respuesta	Nº de DET salida	Tipo de entidad referenciada	Nº de ER[5]	MKII FP

6. *Aplicar la fórmula para calcular FPI.*

   El método MKII utiliza la siguiente fórmula para determinar el *Índice de Puntos Función* o *tamaño funcional* [MKII FPA CPM'98]:

$$FPI = W_I * ON_I + W_E * ON_E + W_O * ON_O$$

   donde W_I, W_E y W_O representan los *pesos promedio en la industria (industry average weights)* para los componentes *entrada* (I), *entidades referenciadas* (E) y

[5] ER: entidad referenciada

de *salida* (O). Sus valores son: $W_I = 0.58$, $W_E = 1.66$ y $W_O = 0.26$ y

N_I: cantidad de *DET* de entrada
N_E: cantidad de *entidades referenciadas*
N_O: cantidad de *DET* de salida

Los *pesos promedio en la industria* suman 2.5 para mantener la correspondencia con los FP de Albrecht. En promedio los métodos dan aproximadamente los mismos tamaños hasta cerca de los 400 FP. Para sistemas más grandes, MKII tiende a producir tamaños mayores que el método de Albrecht [Grant Rule'01[c]].

Hay algunos casos en que es aconsejable obtener los pesos mediante un proceso de calibración. En general, si el objetivo es comparar productividad a través de diferentes entornos, se recomienda usar los *pesos promedio en la industria*. Sin embargo, para estimar dentro de un entorno específico, lo adecuado es obtener los pesos por calibración de proyectos dentro de ese mismo entorno [Symons'91].

7. *Opcional*. En caso de necesitarse calcular los FP considerando los requerimientos técnicos, se debe calcular el *TCA*.

Para el cálculo del factor *TCA*, se utiliza una lista de características generales de la aplicación que se basa en la que usa Albrecht, pero se extiende a 19 características o más si realmente se justifica. Las características adicionales de MKII son:

Características Generales del Sistema			
15	Requerimientos de otras aplicaciones	18	Uso directo por terceras partes
16	Seguridad, privacidad, auditabilidad	19	Documentación
17	Necesidad de entrenamiento al usuario	20	Características definidas por el usuario

Tabla 10 - Características Generales del Sistema adicionales para MKII [Symons'91]

$$TCA = (TDI * C) + 0.65 \text{ [MKII FPA CPM'98]}$$

donde el valor actual promedio en la industria de *C* es 0.005 y *TDI* es el total de los puntajes para cada una de las 19 características, llamado *grado de influencia total*. Los grados de influencia al igual que en el método de Albrecht varían en una escala de cero a cinco, desde ninguna a fuerte influencia [Symons'91].

Luego, el *Índice de Puntos Función Ajustado*, se expresa como:

$$AFPI = FPI * TCA \text{ [MKII FPA CPM'98]}$$

Al comienzo del desarrollo de este punto se ha establecido que es opcional. El *Índice de Puntos Función* mide el *tamaño funcional* de la aplicación desde la visión del usuario, pero el FPA también ofrece un medio para tener en cuenta la complejidad técnica y ciertos requerimientos de calidad, conocidos como requerimientos no funcionales [MKII FPA CPM'98]. Mediante este factor se intenta considerar esos requerimientos. Sin embargo, no hay un acuerdo general acerca de su validez. De hecho en las nuevas versiones del método se considera

que el TCA no contribuye al *tamaño funcional*. La razón es que hay estudios estadísticos que sugieren que este factor no representa en forma adecuada la influencia sobre el tamaño de las características que intenta considerar. En general se podría ignorar cuando se trabaja en un único entorno técnico. Cuando se necesitan comparaciones de tamaño entre aplicaciones construidas con técnicas y requerimientos de calidad muy diferentes puede ser preferible usar un valor de ajuste calculado localmente [MKII FPA CPM'98]. Debe notarse que el valor propuesto para el factor *C* en el cálculo del *TCA* es la mitad del propuesto por Albrecht. La interpretación es que esos factores técnicos actualmente se obtienen con menor esfuerzo que muchos años atrás. Se podría esperar que si esa tendencia continúa, a largo plazo solamente se deberá pensar en los requerimientos de procesamiento de información del problema [Symons'91]. El *TCA* no está incluido dentro de la norma ISO/IEC 14143 y generalmente no se recomienda su uso [MKII FPA CPM'98].

3.7.7. Comparación de los métodos de Albrecht y MarkII [Grant Rule'01[c]]

Este análisis apunta a mostrar las similaridades y diferencias entre las dos variantes del FPA estudiadas. Las técnicas de FPA intentan cuantificar los requerimientos funcionales y no-funcionales. En ambas técnicas esto se logra mediante la determinación inicial del *tamaño funcional* y la posterior aplicación de un factor de ajuste al tamaño, para atender el esfuerzo adicional necesario para cumplir los requerimientos de calidad. Las principales diferencias entre IFPUG (Albrecht) y MKII FPA se encuentran en la forma en que se determina el *tamaño funcional*.

En cualquier técnica de medición del *tamaño funcional* se miden los requerimientos funcionales de una aplicación en términos de uno o más tipos de componentes lógicos, cada uno de los cuales debe tener una relación explícita con el *límite de la aplicación*. Además, se proveen reglas para identificar los tipos de componentes lógicos del software y asignarles un valor numérico que representa su contribución al *tamaño funcional*. Después de identificar los componentes, se determina el *tamaño funcional* de la aplicación mediante la suma del tamaño de cada uno de los tipos de componente lógico.

MKII y IFPUG expresan los requerimientos funcionales en términos de componentes lógicos. MKII usa un único tipo de componente lógico y expresa todos los requerimientos funcionales como *transacciones lógicas*. IFPUG usa cinco tipos de componentes lógicos: *EI*, *EO*, *EQ*, *ILF* y *EIF*.

Los tipos de componentes lógicos están formados por varias partes. En MKII cada *transacción lógica* se compone de entrada, procesamiento y salida. En IFPUG, *EI y EO* se componen de mensajes de entrada y salida, respectivamente, *EQ* por un par entrada/salida, *ILF* y *EIF* por datos persistentes mantenidos por la aplicación u otra aplicación, respectivamente.

En MKII, cada *tipo de entidad* se trata como independiente y se cuentan las *referencias* a las *entidades* por cada *transacción lógica*. En IFPUG, los *tipos de entidad* se agrupan para formar *ILF* si están dentro del *límite de la aplicación* o *EIF* si

están fuera del *límite*. Las *referencias* a los *tipos de entidad* se cuentan como *FTR* por cada *EI*, *EO* o *EQ*.

En ambas técnicas hay una relación entre los tipos de componentes lógicos y el *límite de la aplicación*.

En MKII, por cada *transacción lógica*, el mensaje de entrada y de salida atraviesa el *límite* cuando llega y sale de la aplicación respectivamente y la parte de proceso está completamente retenida dentro del *límite*. En IFPUG, cada *EI* y *EO* atraviesa el *límite* cuando llega y sale de la aplicación, por cada *EQ* la parte de entrada y salida atraviesa el *límite* cuando llega y sale y los *ILF* y *EIF* están completamente retenidos dentro del *límite de la aplicación* que se analiza o de otra aplicación, respectivamente.

Ambas técnicas basan sus reglas para evaluar el tamaño de sus componentes lógicos en las *cuentas base*. En ambas se cuenta la cantidad de *DET* en las partes del mensaje del componente lógico y la cantidad de *referencias* a datos persistentes dentro de la aplicación. Aunque usan diferente terminología, las definiciones son similares.

Mientras en MKII las cuentas resultan de las entradas y salidas de las *transacciones* y de las *referencias* a los datos persistentes, en IFPUG se identifican *ILF* y *EIF* y se cuentan sus partes constituyentes que contribuyen a la funcionalidad.

En MKII se cuenta una *referencia* a una *entidad* por cada *tipo de entidad* accedida durante el curso de una *transacción lógica*, mientras que en IFPUG se cuenta una *FTR* por cada *ILF* o *EIF* accedido durante el curso de una entrada, salida o consulta.

Los objetos de la especificación que se deben identificar son los mismos para ambas técnicas: *mensajes de entrada* y *salida, mensajes de error* y *tipos de entidad*. Además en IFPUG se consideran los *DET* almacenados en los *tipos de entidad*.

Las principales diferencias entre las dos técnicas surgen desde *cómo* se construyen las *cuentas base* y no desde *qué* es lo que se cuenta.

Para determinar las *cuentas base*, se identifican los respectivos componentes lógicos y se cuenta la cantidad de *DET* en los mensajes asociados. Los *mensajes de error* se tratan de manera diferente en las dos técnicas. En IFPUG los atributos de un *mensaje de error* se tratan como atributos de un mensaje de entrada, en MKII se tratan como atributos adicionales de un mensaje de salida. A continuación se cuenta la cantidad de accesos a datos persistentes. En IFPUG, además, se deben ejecutar pasos adicionales para contar la cantidad de *tipos de entidad* en los grupos de *tipos de entidad* que forman cada *ILF* y *EIF* y para contar la cantidad de *DET* almacenados en los *tipos de entidad*.

Hay algunas otras variaciones. MKII usa el concepto de entidad *System* para agrupar las *entidades no primarias* que pueden contener información dependiente de la implementación. IFPUG no cuenta datos dependientes de la implementación.

En MKII las *cuentas base* se usan directamente en los cálculos del *tamaño funcional* expresado en FP sin ajustar. En IFPUG las *cuentas base* se usan para determinar la magnitud de cada componente lógico. Usando las tablas provistas por el

método se valora la magnitud de cada componente como *bajo*, *medio* o *alto*, en base a los valores de las *cuentas base* de *DET* y *referencias* a *tipos de entidad* o la cantidad de *RET* en el caso de *ILF* y *EIF*. IFPUG usa el término *complejidad* en vez de magnitud. No debe confundirse con la *complejidad técnica*.

Los mensajes de *entrada*, *salida* y *referencias* a datos persistentes tienen su propia contribución a una aplicación, pero la contribución de cada una es de diferente clase. Para combinarlas y derivar un único valor numérico del *Índice de tamaño funcional*, se deben normalizar las *cuentas base* para usar una única unidad. Esto se logra a través de los *pesos relativos*.

En MKII, se usan tres pesos: *Wi*, *Wo* y *We*. En IFPUG, el sistema de pesos es más complicado, debido a la gran cantidad de tipos de componentes lógicos. Cada tipo de componente lógico tiene asignado un peso que depende del tipo de componente y su magnitud.

En ambas técnicas el *Índice de tamaño funcional* de la aplicación expresado en FP se obtiene a partir de la suma de las cuentas afectadas por los pesos.

Los dos métodos usan enfoques similares para considerar los requerimientos técnicos. Se evalúa una lista de características no-funcionales en una escala de cero a cinco. IFPUG usa 14 características; MKII usa las mismas 14, pero agrega otras cinco o más.

Una vez evaluado el *grado de influencia* de las características, se suman para dar el *grado de influencia total*. Este valor se usa para ajustar el *Índice de tamaño funcional* y obtener un nuevo valor que representa el *tamaño funcional* y los requerimientos técnicos combinados.