CAPÍTULO 3

Métricas técnicas

Cualquier cosa que queramos medir o predecir en un software es un *atributo* de cualquier *entidad* de un producto, proceso o recurso asociado a éste. Cada entidad de software tiene varios atributos que pueden ser medidos. Es por eso que se necesita hacer una distinción entre atributos que son internos o externos y medidas directas e indirectas:

3.1 Atributos Internos y Atributos Externos

Los *atributos internos* de un producto, proceso o recurso son aquellos que podemos medir puramente en términos del producto, proceso o recurso del mismo. Pueden ser medidos directamente. Por ejemplo: la longitud de un programa o el tiempo transcurrido de cualquier documento de software [Fenton‘91].

Los *atributos externos* de un producto, proceso o recurso son aquellos que solamente pueden ser medidos con respecto al cómo el producto, proceso o recurso se relacionan a su ambiente. Estos tienden a ser los que el administrador y el usuario del software comúnmente gustan de medir y predecir. Por ejemplo el administrador de software le gustaría saber el costo de eficacia de algunos procesos o de la productividad de su personal, mientras los usuarios les gustaría saber la usabilidad, fiabilidad, o portabilidad de un sistema que ellos observan

para comprar. Desgraciadamente los atributos externos son los más difíciles de medir, porque estos no pueden ser medidos directamente [Fenton ’91]. En la tabla 3.1 se describe la estructura, y se dan ejemplos de varios tipos de atributos.

3.2 Medidas Directas y Medidas Indirectas

La *medida directa* de un atributo es aquella, en donde no se depende de cualquier otro atributo. [Fenton´91].

La *medida Indirecta* de un atributo es aquella en la que se involucra la medición de uno o más atributos [Fenton´91].

3.3 El Reto de las Métricas Técnicas

Muchos investigadores han intentado desarrollar una sola métrica que facilite una medida completa de la complejidad del software. Aunque se han presentado docenas de medidas de complejidad, cada una tiene un punto de vista distinto de lo que es la complejidad y de qué atributos de un sistema llevan a la complejidad. Comparemos con una métrica para evaluar un automóvil. Algunos observadores podrían hacer énfasis en el diseño de la cabina, otros podrían hacer hincapié en las características mecánicas, otros podrían considerar el precio, o el rendimiento, o la economía de consumo o la capacidad de reutilizarlo cuando se vaya a desechar. Como cualquiera de estas características puede competir con las otras, es difícil obtener un solo valor del ‘atractivo’ del automóvil. Lo mismo sucede con el software.

Entidades	Atributos	
Recursos	**Internos**	**Externos**
Personal	edad, precio, ..	productividad, experiencia, inteligencia ..
Equipo	tamaño, estructuras ..	productividad, calidad ..
Software	precio, tamaño ..	usabilidad, seguridad ..
Hardware	precio, velocidad, tamaño de memoria, ..	seguridad, ..
Oficinas	tamaño, temperatura, luz, ..	confort, calidad,..
…	…	…
Procesos		
Construcción de especificaciones	tiempo, esfuerzo, número de cambios, ..	calidad, costo, estabilidad
Diseño detallado	tiempo, esfuerzo,..	costo, costo efectivo, ..
Pruebas	número de errores de código encontrados, tiempo, ..	estabilidad, costo, costo efectivo, ..
…	…	…
Productos		
Especificaciones	tamaño, usabilidad modularidad, funcionalidad, ..	comprensibilidad, mantenimiento, ..
Diseños	acoplamiento, modularidad, tamaño, usabilidad, ..	calidad, complejidad, mantenimiento, ..
Código	funcionalidad, complejidad algorítmica, control de flujo	seguridad, usabilidad, mantenimiento
Datos de prueba	tamaño, nivel de protección, ..	calidad,..
…	…	…

Tabla 3.1 Estructura de Atributos [Fenton '91]

Se ha producido una gran cantidad de literatura sobre las métricas del software y es común la crítica de algunas métricas (incluyendo algunas de las presentadas en este documento). Sin embargo, muchas de las críticas se

centralizan en aspectos esotéricos y pierden el objetivo primario de la medición en el mundo real, que es: el ayudar al ingeniero a establecer una manera sistemática y objetiva de conseguir una visión interna de su trabajo y mejorar la calidad del producto como resultado.

Sin embargo sigue existiendo la necesidad de medir y controlar la complejidad del software, y aunque es difícil de adquirir un solo valor de esta "métrica de calidad", sí debería ser posible desarrollar medidas de diferentes atributos internos del programa (por ejemplo: modularidad efectiva, independencia funcional y otros atributos). Las métricas y las medidas conseguidas de ellas pueden usarse como guías independientes de la calidad de los modelos de análisis y de diseño. Pero también surgen problemas aquí. Fenton ['91] lo advierte cuando dice: *'El peligro de intentar encontrar medidas que caractericen tantos atributos diferentes es que inevitablemente las medidas tienen que satisfacer objetivos incompatibles. Esto es contrario a la teoría de representación de la medición".*

Aunque la declaración de Fenton es correcta, mucha gente cuestiona que la medición técnica llevada a cabo en las primeras fases del proceso del software les proporcione a los desarrolladores de software un mecanismo consistente y objetivo para valorar la calidad.

No obstante conviene preguntarse, ¿Qué validez tienen las métricas técnicas?. Es decir, ¿Cómo están relacionadas las métricas técnicas con la fiabilidad y la calidad a largo plazo de un sistema basado en computadora, Fenton['9l] comenta que, a pesar de las conexiones intuitivas entre la estructura interna de los productos de software (métricas técnicas), sus productos externos y

los atributos del proceso, ha habido de hecho, muy pocos intentos científicos para establecer relaciones específicas.

3.4 Identificación del Usuario

En el desarrollo de métricas se requiere identificar claramente al usuario. Lo primero que debemos hacer para identificar a un usuario es preguntarnos:

- ¿Quién necesita la información?
- ¿Quién va a emplear la métrica?
- Si la métrica no tiene un usuario – no utilizarla?

En las preguntas anteriores podemos observar que la selección de una métrica empieza con la identificación de los programas que el usuario quiere medir. ¿Quién necesita la información? Los requerimientos del usuario determinan el programa de métricas. Si el programa de métricas no tiene a un usuario se recomienda no establecer dicho programa.

3.5 Diseño de métricas

Enseguida se planearan los pasos para diseñar métricas: [Lem Ejiogo '91]

Paso 1: Definiciones claras

El primer paso en el diseño de una métrica es el dar una definición estándar para las entidades y sus atributos a ser medidos.

Cuando nosotros empleamos términos como "errores", "defectos", "problema reportado", "tamaño" y aún "proyecto", otra gente podría interpretar

estas palabras en su propio contexto con significados que pueden variar de nuestra intención al definirlos. Estas interpretaciones aumentan sus diferencias cuando se emplean términos más ambiguos como: Calidad, mantenible, amigable al usuario y otros más.

Los ingenieros, administradores u otras personas involucradas en el proyecto pueden emplear diferentes términos para la misma cosa. Por ejemplo, el término defecto reportado, problema reportado, incidente reportado, falla reportada, o reporte de llamada del cliente pueden ser empleados por varias organizaciones para dar significado a la misma cosa. Pero desgraciadamente, pueden también referirse a distintas entidades. Para evitar estos problemas se sugiere:

- Adoptar definiciones estándares y escribirlas
- Aplicarlas con consistencia
- Usar las sugerencias de la industria
- Escoger definiciones que cumplan los objetivos organizacionales

Desafortunadamente, hay muy pocos estándares en la industria para la mayoría de las definiciones de los atributos del software. Cada uno cuenta con una opinión y este debate podría continuar por muchos años. Al abordar este problema se sugieren adoptar definiciones estándares hacia dentro de la organización y aplicarlos consistentemente. Se pueden usar sugerencias de la industria como base para empezar, seleccionar las definiciones que cumplan con los objetivos de la organización y emplearlos para la creación de definiciones propias.

Paso 2: Definir el modelo

El siguiente paso es definir un modelo para la métrica. En términos sencillos, el modelo define el *cómo* calcular la métrica. Por ejemplo, para la Inspección del código de las métricas primitivas, se puede usar los siguientes modelos [Fenton‘91]:

- Líneas de código inspeccionadas = loc
- Horas empleadas en la preparación = hrs-prep
- Medida de preparación = loc / hrs-prep

La mayoría de los modelos incluyen un elemento de simplificación. Cuando nosotros creamos un modelo de medición de software necesitamos ser pragmáticos. Si tratamos de incluir todos los elementos que afectan al atributo o que caracterizan a la entidad el modelo utilizado llegará a ser demasiado complicado. Ser pragmático significa no tratar de crear un modelo perfecto. Tomar los aspectos más importantes. Recordar que el modelo puede ser siempre modificado para incluir mas niveles de detalle en el futuro.

Hay dos métodos para la selección de un modelo:

En muchos casos no es necesario "re-inventar la rueda". Hay muchos modelos de métricas de software que son empleados exitosamente por otras organizaciones. Pero desdichadamente no se saben a puertas abiertas por la “privacidad” que estas requieren.

El segundo método es crear un modelo propio.- El mejor consejo es hablar con la gente quien actualmente es responsable del producto, de los recursos o con

quienes están involucrados en el proceso. Ellos son los expertos y por lo tanto conocen que factores son importantes

Para ilustrar la selección de un modelo, vamos a considerar una métrica para la duración de caídas no planeadas de los sistemas. Si estamos evaluando un sistema de software instalado en un solo servidor, un modelo sencillo tal como los minutos de caída por mes pueden ser suficiente. Si el objetivo es comparar diferentes versiones del software instalado en varios servidores, se puede seleccionar un modelo como minutos de caída por 100 meses de operación. O si queremos enfocarnos en el impacto del cliente, se debe seleccionar minutos de caída al año por "servidor". Se presenta detalladamente, el ejemplo dado de la selección de un modelo [Michael Mah '99].

Duración de caídas no planeadas de sistemas

- Minutos de caídas por mes
- Minutos de caídas / 100 meses de operación
- Minutos de caída por "servidor" por año

Paso 3: Establecer un criterio de conteo

El siguiente paso en el diseño de una métrica es dividir al modelo en sus métricas primitivas (cuantificables en forma directa) y la definición del criterio de conteo para medir cada primitiva. En este paso se definirá el sistema de mapeo para la medición de cada métrica primitiva.

Las métricas primitivas y su criterio de conteo definen el primer nivel de los datos que necesitan ser recolectados para implantar la métrica. Para ejemplificar

esto se va usar el modelo de minutos de caída por servidor por año. Una de las métricas primitivas para este modelo es el número de servidores. Al principio, el conteo de esta primitiva parece sencillo. Pero cuando consideramos la dinámica de añadir nuevos servidores o la instalación de software nuevo en servidores existentes, el criterio de conteo se vuelve más complejo. Así que si empleamos el número de servidores del último día del periodo o ¿calcular un número promedio de servidores para el periodo? De cualquier forma, necesitamos recolectar datos como la fecha en que el sistema fue instalado en el servidor y si queremos comparar diferentes versiones del software, se necesita recolectar la información de las versiones que fueron instaladas en cada servidor y cuando cada una fue instalado. Se presenta detalladamente, el ejemplo dado de criterio de conteo [Michael Mah '99].

- Número de "servidores" al final de un periodo
- Número de "servidores" al inicio + Número de "servidores" al final/ 2 .
- *Ó* ("servidor" en cada día) / número de días

Paso 4: Decidir ¿Qué es bueno?

El cuarto paso en el diseño de una métrica es definir "¿Qué es bueno?". Una vez que se a decidido que medir y como medir, se tiene que decidir que hacer con el resultado. ¿Es 10 demasiado poco o 100 es mucho? . ¿La tendencia debería ser hacia arriba o hacia abajo?

Uno de los primero lugares para empezar a ver "¿Qué es bueno?", es con los clientes. Muchas veces, los requerimientos del usuario determinan ciertos

valores de algunas métricas. Otra fuente de información es la literatura de métricas. Varias investigaciones y estudios han ayudado al establecimiento de normas aceptadas por la industria para mediciones estándares. Si no están disponibles datos históricos, se recomienda esperar hasta recolectar suficiente información para el establecimiento de valores actuales.

Una vez que se ha decidido “¿Qué es bueno?", se pueden determinar en todo caso que acciones son necesarias. Si se está “excediendo” los valores deseados, o si las acciones correctivas no son necesarias. Para establecer el manejo y monitoreo de actividades de mejoramiento, se determinará en las métricas un ambiente en donde se deberá tener presente cuatro metas:

- La meta debe ser razonable; Es correcto el establecer metas que se extienden, pero si ésta es irreal, esta se ignorará.
- La meta debe estar asociada a un marco de tiempo.
- La meta debe fundamentarse en acciones sustentadas. Por ejemplo podría ser razonable establecer una meta de un incremento del 50% en la solución de errores encontrados, si se crea un equipo especial que tendrá como actividad la solución de errores detectados.
- La meta debe ser dividida en partes. Por ejemplo si se emplea un año para alcanzar el mejoramiento, no seria más fácil si la misma meta se divide en 12 pequeñas metas que se establezcan por mes; y así las acciones serán más probables a ser realizadas.

Paso 5: Reporte de la métrica

El siguiente paso es decidir como reportar la métrica. Esto incluye la definición del formato del reporte, la obtención de los datos, mecanismos de reporte de distribución y disponibilidad. El formato del reporte se diseña de acuerdo a lo que se quiera dar a conocer, es por eso que se debe preguntar lo siguiente, ¿Está la métrica incluida en una tabla con otras métricas para un periodo?, ¿Se añade como último valor en una gráfica de tendencia que muestran los valores de la métrica para varios periodos?. Esta gráfica de tendencia ¿Puede ser de barra, línea o de área?, ¿Es mejor comparar valores empleando barras apiladas o gráficas de pastel?. ¿Es mejor hacer tablas y gráficas solas o se agrega texto detallado de análisis y se incluye en el reporte?, ¿Son metas o valores de control los que se incluyen en el reporte?.

En el reporte de métricas, se deberá realizar cuatro pasos: ciclo de obtención de datos, ciclo de reporte de datos, mecanismos de reporte, distribución y disponibilidad que se describen a continuación:

El ciclo de obtención de datos define que a menudo la métrica requiere de snap-shot(s) de los datos y en que momento los elementos de los datos están disponibles para el cálculo de la métrica.

El ciclo de reporte define con que frecuencia el reporte es generado y cuando se prepara para su distribución. Por ejemplo la razón principal de un estudio de métricas puede ser por algún evento, como la terminación de una fase en el proceso de desarrollo del software; Otras métricas como el promedio de defectos reportados pueden obtenerse y reportarse diariamente durante las

pruebas del sistema, la obtención mensual de datos y el reporte mensual después de la liberación del software.

Los mecanismos de reporte proyectan los mecanismos de entrega de las métricas.

Al definir la distribución determina quienes reciben copias del reporte o tienen acceso a la métrica. También aquí se define cualquier restricción al acceso de la métrica, mecanismos de aprobación para añadir y eliminar accesos y cambios en la distribución normal.

Paso 6: Calificadores Adicionales

El paso final en el diseño de una métrica es el determinar calificadores adicionales. Una buena métrica es una métrica genérica. Esto significa que la métrica es válida para todos los niveles de calificadores que se puedan adicionar. Los calificadores añadidos proveen la información estadística necesaria para varios puntos de vista de la métrica. Un ejemplo de calificadores adicionales. [Norman E. Fenton '91] podría ser la duración de caídas no planeadas del sistema, tales como:

- Liberación del producto/ Líneas de productos / Producto final
- Clientes / Segmento de negocios
- Tipo de caída / Causa

En el diseño de las métricas para la duración de caídas no planeadas del sistema pueden emplearse calificadores adicionales para observar toda la información, tanto de las líneas de un producto como, las de productos

individualmente o la liberación del producto, también se pueden percibir las caídas desde el cliente o desde el segmento de negocios, o las podríamos observar por el tipo o causa.

La razón principal de los calificadores adicionales necesitan ser definidos como parte del diseño de las métricas ya que estos determinan el segundo nivel de los requerimientos de la recolección de datos. Ninguna discusión de selección y diseño de métricas de software sería completa sin tener en cuenta el cómo las mediciones afectan a la gente y cómo la gente afecta a las mediciones.

El simple hecho de medir afectara el ambiente de los individuos a ser medidos. Cuando alguien es empezado a ser medido automáticamente asume que tiene importancia. La gente quiere ser vista bien, por lo que van a querer que las mediciones sean buenas. Cuando se crea una métrica siempre se decide que ambientes se desea estimular. Entonces se debe tomar el tiempo necesario de que otros ambientes pueden resultar del empleo o no empleo de métricas.

La mejor manera de prevenir problemas con el factor humano al trabajar con métricas es seguir algunas de las siguientes reglas básicas: [Annelise ‘91]

No hacer mediciones del individuo: Las mediciones de productividad individual son los ejemplos clásicos de estos errores. Si se midiera la productividad en líneas de código por hora producidas, la gente se concentraría en su propio trabajo y excluiría al equipo de trabajo, o se enfocaría en realizar programación con líneas extras de código. Es por eso que se recomienda enfocarse en el proceso y en el producto, no en la gente.

No ignorar los datos: Un camino seguro para acabar un programa de métricas es olvidar los datos cuando se toman las decisiones, ya que estos "dan

sustento a la gente cuando sus reportes emplean información útil a la organización". Si las metas que se establecen y se comunican no son respaldadas con acciones entonces la gente en la organización se desempeñará basandose en el ambiente y no en las metas.

Nunca emplear únicamente una sola métrica: El software es complejo y multifacético es por eso que el enfocarse en una sola métrica puede causar que el atributo medido mejore a expensas de otros atributos. Lo que debería realizarse, es:

Seleccionar las métricas basándose en objetivos: Para tener métricas que cumplan con nuestra necesidad de información, se debe seleccionar métricas que proporcionen información que den respuesta a las preguntas.

Proveer retroalimentación: El proveer retroalimentación con regularidad tiene algunos beneficios:

- El mantener enfocada la necesidad de la recolección de los datos, hará que el equipo vea que los datos actuales son empleados y por lo tanto se volverán consientes de la importancia de la recolección.
- Si los miembros de los equipos son mantenidos con la información de como los datos son usados, ellos serán cada vez menos escépticos de la utilidad de estos.
- Al involucrar a los miembros del equipo en el análisis de los datos y en los esfuerzos de mejoramiento del proceso, el proceso se beneficiara de sus conocimientos y experiencia.

- La retroalimentación en la recolección de datos y en el resultado íntegro ayudará en la responsabilidad de la recolección de los datos, dando como resultado datos más exactos, consistentes y a tiempo.

Obtener "buy-in": Para tener compromiso en las metas como en las métricas, los miembros del equipo necesitan tener un sentimiento de propiedad, es por eso que el participar en la definición de las métricas acrecentara este sentimiento de propiedad. La gente quien trabaja con un proceso a diario tiene un conocimiento intimo del proceso, esto da una perspectiva valiosa de como el proceso se puede medir mejor y como se pueden interpretar los resultados de las mediciones para maximizar su utilización.

3.6 Mediciones del software

Para medir algo se necesita saber que entidades serán medidas y tener una idea de los atributos (propiedades) de la entidad. Primero se debe identificar un atributo y su significado de medición, podemos empezar acumulando datos. Analizando los resultados de estos procesos normalmente permite la clarificación y la re-valuación de los atributos.

3.6.1 Métricas Orientadas al Tamaño

Las métricas del software orientadas al tamaño provienen de la normalización de las medidas de calidad y/o productividad considerando el "tamaño" del software que se haya producido. Si una organización de software mantiene registros sencillos, se puede crear una tabla de datos orientados al tamaño, como la que muestra la figura 3.1, (Pressman ´98) que lista cada proyecto de desarrollo de software y las medidas correspondientes de cada proyecto. Refiriéndonos a la entrada de la figura 3.1 del proyecto *alfa:* se desarrollaron 12,100 líneas de código(LDC) con 24 personas-mes y con un costo de $168,000 dólares. Debe tenerse en cuenta que el esfuerzo y el costo registrados en la tabla incluyen todas las actividades de ingeniería del software (análisis, diseño, codificación y prueba) y no sólo la codificación. Otra información sobre el proyecto alfa indica que se desarrollaron 365 páginas de documentación, se registraron 134 errores antes de que el software se entregara al cliente y se encontraron 29 errores después de entregárselo al cliente dentro del primer año de utilización. También sabemos que trabajaron tres personas en el desarrollo del proyecto alfa.

Proyecto	LDC	Esfuerzo	$(000)	pp. doc.	Errores	Defectos	Personas
alfa	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Figura 3.1 Tabla de datos orientados al tamaño [Pressman '98]

Para desarrollar métricas que se puedan comparar entre distintos proyectos, se seleccionan las líneas de código como valor de normalización. Con los elementales datos contenidos en la tabla se puede desarrollar para cada proyecto un conjunto de métricas simples orientadas al tamaño, tales como:

- errores por KLDC (miles de líneas de código, KiloLDC)
- defectos por KLDC
- $ por LDC
- páginas de documentación por KLDC

Además, se pueden calcular otras métricas interesantes:

- Productividad = KLDC/ persona-mes
- Calidad = errores / KLDC
- Documentación = páginas de documentación / KLDC

Las métricas orientadas al tamaño no están aceptadas universalmente como el mejor modo de medir el proceso de desarrollo del software La mayor parte de la discusión gira en torno al uso de las líneas de código mostradas en la tabla 3.2 (LDC) como medida clave. Los defensores de la medida LDC afirman que la LDC es un “artificio” que se puede calcular fácilmente para todos los proyectos de desarrollo de software, que muchos modelos de estimación del software existente utilizan LDC o KLDC como clave de entrada. En el lado opuesto los ofensores defienden que las medidas en LDC son dependientes del lenguaje de programación, que perjudican a los programas más cortos, pero bien

diseñados, que no pueden acomodar fácilmente lenguajes procedimentales, y que su uso en estimación requiere un nivel de detalle que pude resultar difícil de alcanzar (es decir, el planificador debe estimar las LDC a producirse mucho antes de que se complete el análisis y el diseño)

Lenguaje de Programación	LDC/PF (media)
Ensamblador	320
C	128
Cobol	105
Fortran	105
Pascal	90
Ada	70
Lenguajes Orientados a Objetos	30
Lenguajes de cuarta generación	20
Generadores de código	15
Hojas de cálculo	6
Lenguajes gráficos (iconos)	4

Tabla 3.2 Estimaciones Informales del número medio de LDC [Pressman'98]

3.6.2 Métricas Orientadas a la Función

Utilizan una medida de la funcionalidad; ésta no se puede medir directamente, se debe derivar indirectamente por medio de medidas directas. Las primeras fueron propuestas por Albrecht, que sugirió una medida llamada "*Punto de Función*" para un sistema de software, la idea es que examinemos una especificación del sistema, estas se derivan con una relación empírica según las

medidas contables (directas) del dominio de información del software y las evaluaciones de la complejidad de software. [Charles R. Symons, '98]

El tamaño de la tarea de diseño y desarrollo de un sistema de computo es determinado por el producto de tres factores mostrados en la figura 3.2.

←----------Tamaño "intrínseco del proyecto----------→

(Para estudios de productividad)

El Tamaño de Información Procesada	Factor Técnico de Complejidad	Factores de entorno (medio)
. entradas . salidas . etc.	. batch y en-línea . fácil uso . etc.	. Proyectos administrativos/ Riesgos . Habilidades de las personas, etc. . Métodos, herramientas, lenguajes.

←----------Tamaño total del proyecto ----------→

(Para estimar necesidades)

Figura 3.2 Tamaño de diseño y desarrollo de un sistema de cómputo [Charles R. Symons,'98]

- *El tamaño de información procesada*: éste es una medida de la información procesada y proporcionada por el sistema.
- *Factor técnico de complejidad*: en éste toma en cuenta la medida de varias técnicas y otros factores implicados en el desarrollo y en el implemento de la información procesada requerida.
- *Factores de entorno (o medio*): éste es el grupo de factores que surge del entorno del proyecto típicamente valorado en proyectos con riesgo de medidas. Incluye habilidades, experiencia y motivaciones del personal involucrado y de los métodos, lenguajes y herramientas usadas por el equipo del proyecto.

Nótese que los primeros dos de éstos tres factores son intrínsecos al tamaño del sistema en el sentido que éstos resultan directamente de los requerimientos del sistema que serán entregados al usuario.

El método de Punto de Función ha ganado aceptación en el negocio de sistemas de información, para la evaluación del tamaño del sistema como un componente de la medida de productividad. Cuando están disponibles datos históricos de productividad este método puede también utilizarse como una ayuda a estimar horas-persona. Para estimar propósitos, el tercer grupo de factores del entorno tiene que ser tenido en cuenta también.

El método de Punto de Función de Allan Albrecht consiste en componentes de un sistema que se clasifican en cinco tipos: *entradas externas* (o lógicas), *salidas*, *preguntas*, *interfaces externas a otras sistemas*, y los *archivos lógicos internos*. Dependiendo del número de elementos de datos estos se denominan como “simple”, “promedio” o “complejo”. Cada componente es el número dado de puntos dependiendo en tipo y complejidad (tabla 3.4) y la suma para todos los componentes es expresado en “Puntos funcionales sin ajustar”.

Los factores técnicos de complejidad se determinan, estimando el grado de influencia de algunos componentes “características generales de aplicación” (Tabla 3.3). El grado de influencia en la escala recorre de cero (no presente o no influenciada) hasta 5 (influencia fuerte). La suma de las 14 características (mostradas en la tabla 3.3), que es el *Grado Total de Influencia* (**DI**), se convierte al *Factor Técnico de Complejidad* (**TCF**) calculándose:

$$\mathbf{TCF} = 0.65 + 0.01 * \quad Di \qquad (3.1)$$

El valor de Di, donde los valores de ajuste de complejidad *i* va de 1 a 14 según las respuestas de las preguntas de tabla 3.3:

Di	Preguntas
C1	¿ Requiere el sistema copias de seguridad y de recuperación fiables?
C2	¿ Se requiere de comunicación de datos?
C3	¿ Existen funciones de procesamiento distribuido?
C4	¿ Es crítico el rendimiento?
C5	¿ Se ejecutará el sistema en un entorno operativo existente y fuertemente utilizado?
C6	¿ Requiere el sistema entrada de datos interactiva?
C7	¿ Requiere la entrada de datos interactiva que las transacciones de entrada se lleven a cabo sobre múltiples pantallas u operaciones?
C8	¿ Se actualizan los archivos maestros de forma interactiva?
C9	¿ Son complejas las entradas, salidas, archivos o las peticiones?
C10	¿ Es complejo el procesamiento interno?
C11	¿Se ha diseñado el código para ser reutilizable?
C12	¿ Están incluidas en el diseño la conversión y la instalación?
C13	¿Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes
C14	organizaciones?
	¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?

Tabla 3.3 Preguntas de Di [Charles R. Symons, ´98]

Valores de Di

No presente o no influencia = 0

Influencia insignificante o incidental = 1

Influencia moderada = 2

Influencia promedio o medio = 3

Influencia significante = 4

Influencia esencial o fuerte, a través de = 5

Los valores constantes de la ecuación anterior (3.1) y los pesos que se aplican a las cuentas de los dominios de información se determinan empíricamente.

El tamaño intrínseco relativo del sistema en Puntos Funcionales (**FP´s**) se calcula con ayuda de la tabla 3.4, utilizando la siguiente fórmula:

$$\textbf{FP´s} = \text{UFP´s} * \text{TCF} \qquad (3.2)$$

Descripción	Simple	Promedio	Complejo	Total
Entradas externas	_ * 3 = _	_ * 4 = _	_ * 6 = _	__
Salidas externas	_ * 4 = _	_ * 5 = _	_ * 7 = _	__
Archivos internos lógicos	_ * 7 = _	_ *10 = _	_ *15 = _	__
Archivos de interfaz externa	_ * 5 = _	_ * 7 = _	_ *10 = _	__
Indagación externas	_* 3 = _	_ * 4 = _	_ * 6 = _	__
			(**UFP**) Total de Puntos funcionales sin ajustar = __	

Tabla 3.4 Nivel de Información Procesando Funciones [Pressmna'98]

Podemos notar que los Puntos Funcionales son por lo tanto números dimensiónales en una escala arbitraria.

Los valores de la información mostrados en la tabla 3.4 se definen a continuación:

- *Entradas Externas* (o número de entradas de usuario). Se suma cada entrada dada por el usuario, donde nos proporcione distintos datos orientados a la aplicación. Estas se diferencian de las peticiones.

- *Salidas Externas* (o número de salidas de usuario*).* Se suma cada salida que le proporcionará al usuario información orientada a la aplicación (informes, pantallas, mensajes de error, etc.). Los elementos de datos particulares de un informe no se cuentan de forma separada.
- *Archivos Internos Lógicos o Número de archivos.* Se suma cada archivo maestro lógico (grupo lógico de datos que sean parte de una base de datos o un archivo independiente).
- *Archivos de Interfaz Externa o Número de interfaces externas.* Se suman todas las interfaces legibles por la máquina (archivos de datos de cinta o discos, etc.) que se utilizan para transmitir información a otro sistema.
- *Indagaciones externas o Número de peticiones de usuario.* La petición es una entrada dada que nos va a producir una respuesta inmediata del software en forma de salida. Las peticiones se cuentan por separado.
- *Total de Puntos funcionales sin ajustar o Cuenta-Total.* Es la suma de todas las entradas obtenidas de la tabla 3.4

Cuando se calculan los puntos de función, éstos se utilizan de forma análoga a las LDC (Líneas de Código) para normalizar medidas de productividad, calidad y otros ámbitos de software, como por ejemplo:

- Errores por Puntos de Función.
- Defectos por Puntos de Función.
- Costo (dinero) por Puntos de Función.
- Página de documentación por Puntos de Función.
- Puntos de Función por persona-mes.

Las razones de Albrecht para proponer los Puntos Funcionales como medidas de tamaño de un sistema son: [Pressman ‘98]

- Estas medidas aíslan el tamaño intrínseco del sistema de los factores del medio, facilitando el estudio de factores que influyen en la producción.
- Estas medidas están basadas; en las observaciones de los usuarios externos del sistema, y es tecnología independiente.
- Estas medidas pueden determinarse al inicio del ciclo de desarrollo lo que permite utilizar los Puntos Funcionales en la estimación de procesos.
- Los Puntos Funcionales pueden ser entendidos y evaluados por usuarios que no son técnicos.

3.6.3 Medidas de complejidad de Halstead

Las medidas fueron desarrolladas por Maurice Halstead para determinar una medida cuantitativa de la complejidad directa de los operarios y operandos en el módulo. Entre las métricas de software más avanzadas, éstos son indicadores fuertes de complejidad de código; se basa ampliamente en la evidencia empírica encontrada en el trabajo del “Maintainability Index Work”, pero existen la evidencia de que las medidas de Halstead son también útiles durante el desarrollo, para valorar la calidad de código en aplicaciones computablemente-densas.

Las medidas de Halstead se basan en cuatro números de escalar, que serán derivados directamente del código de fuente del programa, tales como:

n1 = el número de distintos operador

n2 = el número de distintos operandos

N1 = el número total de operandos

N2 = el número total de operador

De estos números, cinco medidas se derivan (tabla 3.5)

Medidas	Símbolo	Formula
Longitud del programa	N	N= N1 + N2
Vocabulario del Programa	n	n= n1 + n2
Volumen	V	V= N * (LOG2 n)
Dificultad	D	D= (n1/2) * (N2/n2)
Esfuerzo	E	E= D * V

Tabla 3.5 Fórmulas derivadas de los números de escalar de Halstead [Lem O. Ejiogo '91]

3.7 Paradigma Meta/ Pregunta/ Métrica

El Paradigma Meta/Pregunta/Métrica (Goal-Question–Metric), provee un mecanismo excelente para la definición de un programa de medición basado en metas. Funciona de la siguiente manera [Marty A. '90]:

- El primer paso es definir una o más metas medibles. Éstas pueden ser metas estratégicas de un alto nivel, como minimizar el costo o maximizar la satisfacción del usuario. Se puede especificar metas como la evaluación de la efectividad de procesos nuevos o determinar si un producto está listo para ser liberado al usuario.

- El segundo paso es definir las preguntas necesarias para ser contestadas, y determinar si la meta es cumplida.
- El paso final es determinar que métricas son necesarias para contestar cada pregunta.

3.8 Ciclo del tiempo

El ciclo de tiempo es una medida que nos proporciona el tiempo se emplea para llevar a cabo un proceso. Hay dos medidas de tiempo: [John A. ‘91]

- Ciclo de tiempo estático se utiliza el tiempo promedio actual que se emplea para ejecutar el proceso. Por ejemplo que tiempo emplea un módulo en corregir una falla y ejecutar un caso de prueba.
- Ciclo de tiempo dinámico es calculado al dividir el número de elementos en progreso (elementos que únicamente han completado el proceso parcialmente) entre los nuevos elementos comenzados y nuevos elementos terminados durante el periodo.

Conociendo el ciclo del tiempo para los subprocesos en el proceso de desarrollo de software se podrá hacer una mejor estimación del plan y de los recursos requeridos, también nos permite monitorear el impacto de las actividades del proceso en el mejoramiento del ciclo de tiempo para este proceso.

El siguiente ejemplo ilustra el cálculo de métricas del ciclo de tiempo estático y del ciclo de tiempo dinámico:

Ciclo del tiempo estático:

Calendario de tiempo para completar el proceso:
Módulo A: 5 días
Módulo B: 10 días
Módulo C: 7 días
Módulo D: 8 días
(5+10+7+8)/4 = 7.5 días

Ciclo de tiempo dinámico:

Total de elementos en progreso / (elementos iniciados + elementos completados)/ 2 (3.5)

52 módulos iniciados en este mes
68 módulos terminados en este mes
12 módulos en progreso en este mes
(12/ ((52+68) /2)) * 30 días en el mes = 6 días

3.9 Diferentes enfoques de Métricas

La relación entre las líneas de código y los puntos de función depende del lenguaje de programación que se utilice para implementar el software y de la calidad del diseño. Las medidas LDC y PF se utilizan a menudo para extraer métricas de productividad. Esta invariabilidad conduce a la discusión sobre el uso de tales datos. Se debe comparar la relación LDC/personas-mes (o PF/PM) de un grupo con los datos similares de otro grupo?, ¿Deben los administradores evaluar el rendimiento de las personas usando estas medidas? La respuesta a estas preguntas es un terminante “No”. La razón para esta respuesta es que hay muchos factores que influyen en la productividad, haciendo que la comparación de “peras y manzanas” sea mal interpretada con facilidad. Basili y Zelkowitz definen

cinco factores importantes que inciden en la productividad del software [Charles R. Symons´91]:

- *Factores humanos.* El tamaño y la experiencia de la organización de desarrollo.
- *Factores del problema.* La complejidad del problema que se debe resolver y el número de cambios en las restricciones o los requisitos del diseño.
- *Factores del Proceso.* Técnicas del análisis y diseño que se utilizan, lenguajes y herramientas CASE y técnicas de revisión.
- *Factores del producto.* Fiabilidad y rendimiento del sistema basado en computadora.
- *Factores del recurso.* Disponibilidad de herramientas CASE, y recursos de hardware y software.

Si uno de los factores de productividad está por encima de la media (altamente favorable) para un proyecto dado, la productividad de desarrollo del software será significativamente más alta que el mismo factor por debajo de la media (desfavorable).

3.10 Recursos, Procesos y Productos

La primera obligación en cualquier actividad de medición de software es el identificar las entidades y atributos de interés que deseamos medir. Sabiendo de antemano que una *entidad* es un objeto o un evento y un *atributo* son las características o propiedades del software a medir. En el software hay tres clases

de entidades cuyos atributos vamos a medir y se muestra en la figura 3.3 [Anneliese von M. '91.

Figura 3.3 Entidades del Software [Anneliese von M. '91]

Recursos: son los artículos que corresponde a las entradas a procesos.

***Procesos*:** es cualquier software relacionado con las actividades, teniendo éstos normalmente un factor de tiempo.

***Productos*:** cualquier artefacto; liberados o documentos en el cuál surja afuera de los procesos.

3.10.1 Recursos

Los recursos son los diversos puntos de partida a considerar para la producción de software. La estimación de los recursos requeridos para acometer el esfuerzo de desarrollo de software se muestra en la figura 3.4 en forma de pirámide. En la base de la pirámide de recursos se encuentra el *entorno de desarrollo* – hardware y software que nos van a proporcionar la infraestructura de soporte al esfuerzo desarrollo. En un nivel más alto se encuentran los *componentes de software reutilizables* – que son bloques de software que pueden reducir drásticamente los costos de desarrollo y acelerar la entrega. En la parte más alta de la pirámide está el *recurso primario*: las personas.

Cada recurso queda especificado mediante cuatro características: descripción del recurso, informe de disponibilidad, fecha cronológica en la que se requiere el recurso, tiempo durante el que será aplicado el recurso. Las dos últimas características pueden verse como una *ventana temporal*. La disponibilidad del recurso para una ventana específica tiene que establecerse lo más pronto posible [Pressman '91].

Recursos Humanos: El encargado de la planificación empezará elevando el ámbito y seleccionado las habilidades técnicas requerida para llevar a cabo el desarrollo. Dentro de la selección se deberá especificar la posición que ocupará dentro de la organización (ingeniero de software, administrador,...) y la especialidad (telecomunicaciones, bases de datos, microprocesadores, administrador de proyectos o de grupo). Para proyectos relativamente pequeños (una persona-año o mes o menos) una sola persona puede llevar a cabo todos los pasos de ingeniería del software, consultado con especialistas siempre que requiera. El número de personas requerido para un proyecto de software sólo puede ser determinado después de hacer una estimación del esfuerzo de desarrollo (personas-mes, personas-año).

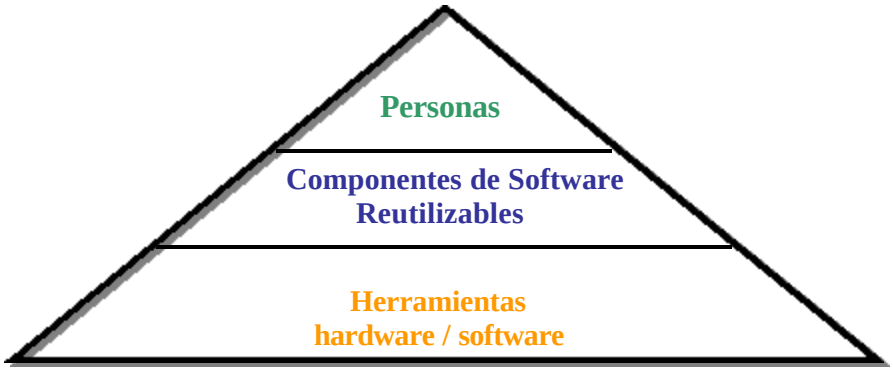

Figura 3.4 Recursos de Desarrollo de Software [Pressmna '98]

Recurso de software reutilizable: Se refiere la creación y reutilización de bloques de construcción de software, en donde deberán establecerse en catálogos para una consulta más fácil, estandarizarse para una fácil aplicación y validarse para la también una fácil integración. Pressman [´98] sugiere cuatro categorías de recursos de software que se deberán tener en cuenta a medida que se avanza con la planificación:

Componentes ya desarrollados: El software ya existente se puede adquirir de una tercera parte o provenir de un desarrollo internamente para un proyecto anterior. Por lo tanto estos componentes están listos para utilizarse en el proyecto actual y se han validado totalmente.

Componentes ya experimentados: Son las especificaciones, diseño, código o datos de prueba existentes y desarrollados para proyectos anteriores que son similares al software que se va a construir para el proyecto actual y los miembros del equipo de software actual ya han tenido la experiencia completa en el área de la aplicación representada para estos componentes. Y por lo tanto las modificaciones requeridas tendrán un riesgo relativamente bajo.

Componentes con experiencia parcial: Las especificaciones, los diseños, código o los datos de prueba existentes desarrollados para proyectos anteriores que se relacionan con el software que se va a construir para el proyecto actual, pero que requerirán una modificación sustancial y los miembros del equipo de software actual han limitado su experiencia sólo al área de aplicación representada por estos componentes y es por eso que

las modificaciones requeridas para componentes de experiencia parcial tendrán un mayor grado de riesgo.

Componentes nuevos. Los componentes de software que el equipo debe construir específicamente para las necesidades del proyecto actual.

De forma irónica, a menudo se descuida la utilización de componentes de software reutilizables durante la planificación, llegando a convertirse en la preocupación primordial durante la fase de desarrollo del proceso de software. Por lo tanto, es mejor especificar al principio las necesidades de recursos del software. De esta forma se puede dirigir la evaluación técnica de alterativas y puede tener lugar la adquisición oportuna.

Recursos de entorno. El entorno es en donde se apoya el proyecto de software, llamado a menudo *entorno de ingeniería de software* (EIS), que incorpora hardware y software, en donde el hardware aporta una plataforma con las herramientas (software) requeridas para producir los productos finales.

Cuando a un sistema basado en computadora (hardware y software especializado) le es aplicado ingeniería, el equipo de software puede requerir acceso a los elementos en desarrollo por otros equipos de ingeniería y es por eso que cada elemento de hardware debe ser especificado por el planificador del proyecto de software.

Un atributo de gran interés el cuál es relevante para todos, es el *"costo,* que es considerado en muchas situaciones, dependiente del número de atributos en adición a los cuales son mas fáciles de medir, llamado *precio*. En el caso de personas como un recurso en adición al costo, será de interés el atributo de *producción.* Este será externo porque es dependiente en un proceso en particular.

Otros atributos de interés de personas individuales son *experiencia, edad, inteligencia* o, como equipos *tamaño, estructura, y tipos de comunicación*" [Fenton ´91].

3.10.2 Procesos

Las entidades de procesos de software incluyen actividades relacionadas con el software y eventos que usualmente son asociados con un factor de tiempo. Las métricas del proceso de software se utilizan para propósitos estratégicos. Por ejemplo: actividades definidas, como el desarrollo de un sistema de software desde los requerimientos hasta la liberación del usuario o la inspección de una parte del código. Las métricas de proceso también se extraen midiendo las características de tareas específicas de la ingeniería de software y obteniendo como resultados medidas de errores detectados antes de la entrega del software, defectos detectados e informados por los usuarios finales, productos de trabajo entregados, el esfuerzo humano y tiempo consumido, ajuste con la planificación y otras medidas. Por ejemplo: [Fenton´91]

Ejemplo 1: Quizás sea razonable usar una medida indirecta para un proceso como una prueba formal:

$$\frac{Costo}{Número\ de\ errores\ encontrados} \qquad (3.3)$$

Esta fórmula nos da el promedio entre el costo y el número de errores encontrados durante el proceso, en donde el costo serían los atributos de

procesos externos los cuales se cree que son importantes tales como: *controlabilidad, observabilidad y estabilidad*

Ejemplo 2: Se podría usar la fórmula 3.4 como la base para una medida de la estabilidad del proceso del diseño durante un período específico de tiempo. Específicamente:

$$\textit{estabilidad de diseño} = \frac{\textit{número total de metodologías de diseño}}{\textit{número de procesos del diseño}} \qquad (3.4)$$

Los indicadores de procesos permiten a una organización de ingeniería del software tener una visión profunda de la eficacia de un proceso ya existente; por ejemplo: el paradigma, las tareas de ingeniería del software, los productos del trabajo e hitos. También permiten que los administradores evalúen lo que funciona y lo que no. Las métricas de proceso se recopilan de todos los proyectos y durante un largo periodo de tiempo. El objetivo es proporcionar indicadores que lleven a mejoras de los procesos de software a largo plazo.

Según Pressman [‘98] los indicadores de proyecto permiten al administrador de software:

- Evaluar el estado del proyecto en curso.
- Seguir la pista de los riesgos potenciales.
- Detectar las áreas de problemas antes de que se conviertan en “críticas”
- Ajustar el flujo y las tareas de trabajo.
- Evaluar la habilidad del equipo del proyecto en controlar la calidad de los productos de trabajo de la ingeniería del software.

La única forma racional de mejorar cualquier proceso es medir atributos del proceso, desarrollando un juego de métricas para proporcionar indicadores que conducirán a una estrategia de mejora, en la figura 3.5, se muestra al *proceso,* en el centro de un triángulo que es conectado por tres factores con una gran influencia en la calidad del software y en el rendimiento como organización donde la complejidad del *producto* puede tener un impacto sustancial sobre la calidad y el rendimiento del equipo, la *tecnología* (p. Ej. métodos de la ingeniería del software) que puebla el proceso también tiene su impacto. Están dentro del círculo de condiciones entornos que incluyen “Entornos de Desarrollo” (p. Ej. herramientas CASE), “Condiciones del Negocio” (p. Ej. fechas, tope, reglas de empresa), y “Características del Cliente” (p. Ej.: facilidad de comunicación”.

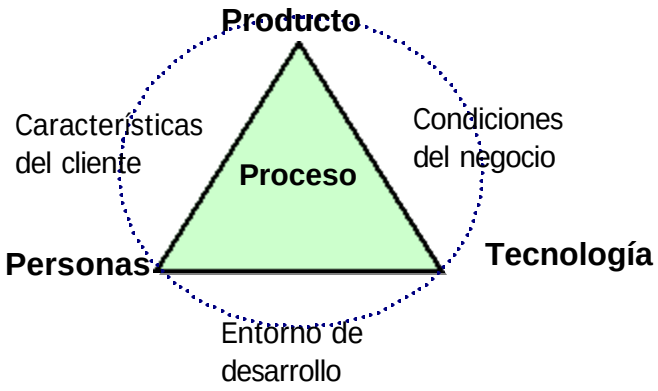

Figura 3.5 Proceso [Pressman ‘98]

Mayhauser [’91] argumenta que existen usos “privados y públicos” para diferentes tipos de datos del proceso. Los ingenieros de software podrán sentirse no familiarizados con la utilización de métricas recopiladas de una base particular, estos datos deberían ser *privados* para el individuo y servir sólo como un indicador de ese individuo, algunos ejemplos podrían ser los índices de defectos, los índices

de defectos por módulo, errores encontrados durante el desarrollo, ya que sabemos que los datos *privados* de proceso pueden servir como referencia importante para mejorar el trabajo individual del ingeniero de software.

Mayhauser [’91] menciona que algunas métricas de proceso son *privadas* para el equipo del proyecto de software, pero *públicas* para todos los miembros del equipo. Entre los ejemplos se incluyen los defectos informados de funciones importantes del software (que un grupo de profesionales han desarrollado), errores encontrados durante revisiones técnicas formales y líneas de código o puntos de función por módulo y función. El equipo revisa los datos para detectar los indicadores que pueden mejorar el rendimiento del equipo.

Las métricas de proceso del software pueden proporcionar grandes resultados a medida que una empresa trabaja para aumentar su nivel de madurez del proceso. También favorecen a clarificar el estatus de los proyectos y su aprobación con el plan de desarrollo ya que al crear un buen proceso durante el desarrollo del software (buenas técnicas, administración, horarios, etc.) se reflejará en la calidad de desarrollo del software. Aunque como en todas las métricas éstas pueden ser mal utilizadas originando así un mayor número de problemas, Robert U. Charette [’88] sugiere una etiqueta de métricas del software apropiadas para administradores a medida que instituyen un programa de métricas de proceso:

- Utilice el sentido común y una sensibilidad organizativa al interpretar datos de métricas.
- Proporcione una realimentación regular a particulares y equipos que hayan trabajado en la recopilación de medidas y métricas.
- No utilice métricas para evaluar a particulares

- Trabaje con profesionales y equipos para establecer objetivos claros y métricas que se vayan a utilizar para alcanzarlos.
- No utilice nunca métricas que amenacen a particulares o equipos.
- Los datos de métricas que indican un área de problemas no se deberían considerar “negativos”. Estos datos son meramente un indicador de mejora del proceso.
- No se obsesione con una sola métrica y excluya otras métricas importantes.

A medida que una organización está más a gusto con la recopilación y utilización de métricas de proceso, la derivación de indicadores simples abre el camino hacia un enfoque más riguroso, llamado M*ejora de Estadística del Proceso del Software* **MEPS** (Statistical Software Process Improvement (SSPI)), Pressman [’98] menciona que MEPS utiliza el análisis de fallas del software para recopilar información de errores y defectos encontrados en el desarrollo y utilizar una aplicación de sistema o producto. El análisis de fallos funciona de la siguiente manera:

- Todos los errores y defectos se categorizan por origen (p. ej. : defectos en la especificación, en la lógica, no acorde con los estándares).
- Se registra tanto el costo de corregir cada error como el del defecto.
- El número de errores y de defectos de cada categoría se cuentan y se ordenan en orden descendente.
- Se calcula el costo global de errores y defectos de cada categoría que producen el costo más alto para la organización.
- Los datos resultantes se analizan para detectar las categorías que producen el costo más alto para la organización.

Se desarrolla planes para modificar el proceso con el intento de eliminar (o reducir la frecuencia de apariciones) la clase de errores y defectos que sean más costosos.

3.10.2.1 Métricas del proyecto.

Las medidas del proyecto de software son tácticas. Esto es, las métricas de proyectos y los indicadores derivados de ellos los utilizan un administrador de proyectos y un equipo de software para adaptar el flujo de trabajo del proyecto y las actividades técnicas.

La primera aplicación de métricas de proyectos en la mayoría de los proyectos de software ocurre durante la estimación. Las métricas recopiladas de proyectos anteriores se utilizan como una base desde la que se realizan las estimaciones del esfuerzo y del tiempo para el actual trabajo de software. A medida que avanza un proyecto, las medidas del esfuerzo y del tiempo consumido se comparan con las estimaciones originales (y la planificación del proyecto). El administrador de proyectos utiliza estos datos para supervisar y controlar el avance.

A medida que comienza el trabajo técnico, otras medidas de proyeclos comienzan a tener significado. Se miden los índices de producción representados mediante páginas de documentación, las horas de revisión, los puntos de función y las líneas fuente entregadas Además, se sigue la pista de los errores detectados durante todas las tareas de ingeniería del software. Cuando va evolucionando el software desde la especificación al diseño, se recopilan las métricas técnicas

(Capítulo 3) para evaluar la calidad del diseño y para proporcionar indicadores que influirán en el enfoque tomado para la generación de código, módulos y pruebas de integración.

La utilización de métricas para el proyecto tiene dos aspectos fundamentales [McDermid' 91]: En primer lugar, estas métricas se utilizan para minimizar la planificación de desarrollo guiando los ajustes necesarios que eviten retrasos y atenúen problemas y ríesgos potenciales. En segundo lugar, las métricas para el proyecto se utilizan para evaluar la calidad de los productos en el momento actual y cuando sea necesario, modificar el enfoque técnico del mejoramiento de la calidad.

A medida que mejora la calidad, los errores se minimizan y el número de defectos disminuyen, la cantidad de trabajo que ha de rehacerse también se reduce Esto lleva a una reducción del costo global del proyecto.

Otro modelo de métricas del proyecto de software sugiere que todos los proyectos deberían medir: [John A. Mayhauser '91]

- Entradas: la dimensión de los recursos (p. ej personas, medio ambiente) que se requieren para realizar el trabajo.
- Salidas. medidas de las entregas o productos creados durante el proceso de ingeniería del software
- Resultado. medidas que indican la efectividad de las entregas

En realidad, este modelo se puede aplicar tanto al proceso como al proyecto En el contexto del proyecto, el modelo se puede aplicar recursivamente a medida

que aparece cada actividad estructural. Por consiguiente, las salidas de una actividad se convierten en las entradas de la siguiente. Las métricas de resultados se pueden utilizar para proporcionar una indicación de la utilidad de los productos cuando fluyen de una actividad (o tarea) a la siguiente.

3.10.2.2 Integración de las métricas dentro del proceso de software

La mayoría de los desarrolladores de software todavía no miden, y por desgracia, la mayoría no desean ni comenzar. Como se ha señalado, el problema es cultural. En un intento por recopilar medidas en donde no se haya recopilado nada anteriormente, a menudo se opone resistencia: "¿Por qué necesitamos hacer esto?", se pregunta un administrador de proyectos fatigado. "No entiendo por qué", se queja un profesional saturado de trabajo. ¿Por qué es tan importante medir el proceso de ingeniería del software y el producto (software) que produce? La respuesta es relativamente obvia. Si no se mide, no hay una forma real de determinar si se está mejorando, y si no se está mejorando, se está perdido [Pressman '93]. La medición es una de las "medicaciones" que pueden ayudar a curar el "mal del software". Esta proporciona beneficios al nivel de proyecto, estratégico y técnico.

La administración de alto nivel puede establecer objetivos significativos de mejora del proceso de ingeniería del software solicitando y evaluando las medidas de productividad y de calidad. Se señaló que el software es un aspecto de administración estratégico para muchas compañías. Si el proceso por el que se desarrolla puede mejorarse, entonces puede producirse un impacto directo en lo

sustancial. Pero para establecer objetivos de mejora, se debe comprender el estado actual de desarrollo del software.

Los rigores del trabajo diario de un proyecto del software no dejan mucho tiempo para pensar en estrategias. Los administradores del proyecto de software están más preocupados por aspectos mundanos (aunque igualmente importantes): desarrollo de estimaciones significativas del proyecto, producción de sistemas de alta calidad, y terminar el producto a tiempo. Mediante el uso de la medición para establecer una línea base del proyecto, cada uno de estos asuntos se hace más fácil de manejar. Ya se ha apuntado que la línea base sirve como un lineamiento para la estimación. Además, la recopilación de métricas de calidad permite a una organización "sintonizar" su proceso de ingeniería del software para eliminar las causas "poco vitales" de los defectos, que tienen el mayor impacto en el desarrollo del software.

Técnicamente (en las trincheras) las métricas del software, cuando se aplican al producto, suministran beneficios inmediatos. Cuando se ha terminado el diseño del software, la mayoría de los que desarrollan pueden estar ansiosos por obtener respuestas a preguntas como:

- ¿Qué requisitos del usuario son más susceptibles al cambio?
- ¿ Qué módulos del sistema son más propensos a error?
- ¿Cómo se debe planificar la prueba para cada módulo?
- ¿Cuántos errores (de tipos concretos) puede esperar cuando comience la prueba?

Se pueden encontrar respuestas a esas preguntas si se han recopilado métricas y se han usado como guía técnica.

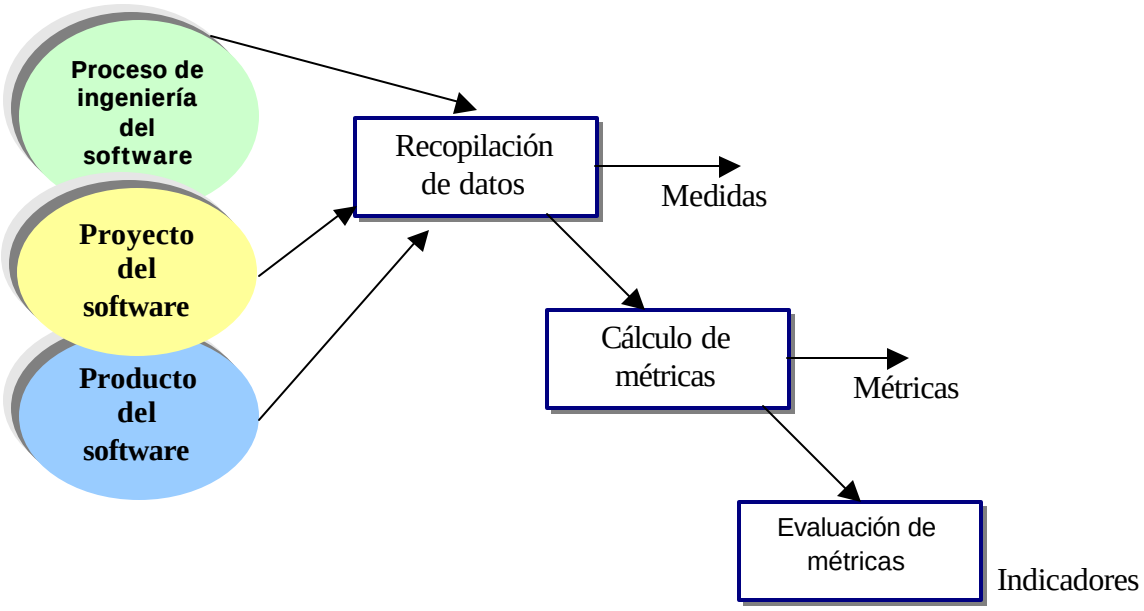

Figura 3.6 Proceso de recopilación de métricas de software [Pressman '98]

El proceso que se establece en una línea base (datos recopilados de proyectos de desarrollo de software anteriores) se muestra en la figura 3.6. Idealmente, los datos necesarios para establecer una línea base han sido recopilados a medida que sé ha ido progresando. Por desgracia, este no es el caso. Por consiguiente, la *recopilación de datos* requiere una investigación histórica de los proyectos anteriores para reconstruir los datos requeridos, una vez que se han recopilado medidas (el paso más difícil), el *cálculo de métricas* es posible. Dependiendo de la amplitud de las medidas recopiladas, las métricas pueden abarcar una gran gama de métricas, tales como: LDC y PF, así como métricas de calidad y orientadas a objetos. Finalmente, las métricas se deben

evaluar y aplicar durante la estimación, el trabajo técnico, el control del proyecto y la mejora de proyectos, la *evaluación de métricas* se centra en las razones de los resultados obtenidos, y produce un grupo de indicadores que guían el proyecto o el proceso.

3.10.3 Productos

Los productos del software son las salidas del proceso de producción del software. Éstas incluyen todos los artefactos entregados o documentos que son productos durante el ciclo de vida del software. Anneliese Von [‘91] menciona que las métricas de los productos usualmente cuantifican algunos aspectos de calidad relacionados a la tabla 3.6.

Lista de Requerimientos de Calidad	
Las capacidades de los productos	El grado de la funcionalidad
Usabilidad	fiabilidad
desempeño (eficiencia)	seguridad
factores humanos	Portabilidad
La extensión de la solución	Futuro/expectativas del tiempo de vida
Compatibilidad	Mantenibilidad:
Adaptabilidad	Correctivo
Recursos	Perfectivo
Limitaciones personales	Instalación
Limitaciones presupuestales	Horario
Beneficios:	Limitaciones físicas
Tangibles	Documentación
Intangibles	Limitaciones organizacionales

Tabla 3.6 Aspectos de calidad de software

No siempre será posible medir éstos directamente, especialmente cuando las métricas son *predecibles*, por ejemplo, cuando el software no esta en desarrollo y la métrica es usada para predecir su calidad operacional; entonces los subfactores y atributos deliberados son medidas que correlacionan dando como resultado una métrica indirecta.

3.10.3.1 Modelo de la Planificación Predictiva

Es intuitivamente obvio que un programa largo tiende a ser más complejo y más costoso a desarrollar y mantener. Un programa largo también tiende a tener un vocabulario grande y usa más operaciones y operandos. También tiene más líneas de código (LOC). La figura 3.7 nos muestra una relación entre métricas y el modelo de desarrollo para las variables que planean. El signo "?" En la fórmula nos indica un factor de tamaño en donde puede estar influyendo en los parámetros de la salida del modelo. Este modelo de fórmulas sólo trabaja si el tamaño tiene una crucial influencia en el tiempo, costo y requisitos de los recursos para un proyecto y si la relación entre tamaño y complejidad es fuerte. Si este no es el caso, el modelo quizá trabaje para algún proyecto pero no para otros.

Cuando el tamaño no es el único contribuidor significante a la variable planeada, el modelo a predecir entonces será multidimensional y más complejo.

Un *modelo* es una relación de la forma Y = *f*(*x1, x2, . . ., xm).* Y es la variable que deseamos predecir. Esta es llamada *variable dependiente.* La *X*i son las *variables independientes* en que la predicción depende de Y. Mayrhauser menciona que hay dos caminos para establecer esta relación funcional:

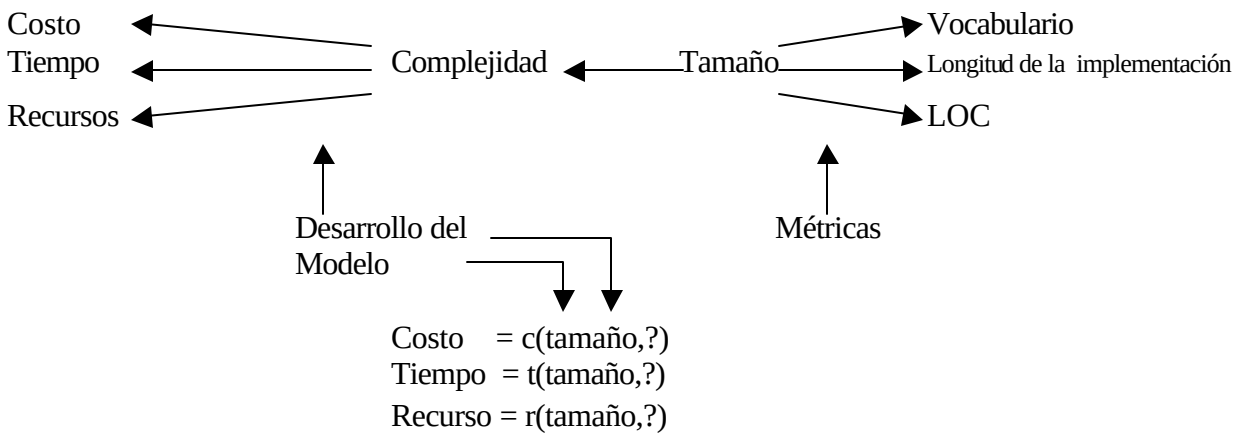

Figura 3.7 Métricas Posibles para un modelo predictivo [Fenton ´91]

El primero es, por análisis estadístico de los datos del caso. Este es un *modelo empírico.* Muchos modelos de costo de estimación son de este tipo, considerándose son tan representativos como los datos sobre los que estos son basados y la calidad de los métodos estadísticos de la inferencia que se usan.

El segundo camino se lleva acabo a través de razonamiento analítico y pruebas que establecen la relación entre variables independientes y dependientes. Algunos modelos analíticos postulan una familia de funciones con situaciones constantes. Estos tienen que ser medidos separadamente o estimados por métodos estadísticos. El modelo se *calibra* a su medio (o ambiente) en la aplicación.