

GUIA DE PREGUNTAS

Material "PROCESO DE SOFTWARE Y METRICAS DE PROYECTO.
Ingeniería del Software (Capítulo 4) de Roger S. Pressman (5^{ta} Edición)"

1. Defina en el contexto de la Ingeniería de Software: Medida, Medición, Métrica e Indicador.
2. Defina indicador de Proceso.
3. Defina Indicador de Proyecto
4. Enuncie el programa de métricas de proceso propuesto por Grady.
5. Enuncie las buenas prácticas del análisis de fallas.
6. Defina Métricas orientadas al Tamaño y Métricas Orientadas a la Función. De Ejemplos.
7. Defina Puntos de Función.
8. Enuncie las características del dominio de información tenidas en cuenta en el cálculo de puntos de función.
9. Describa el cálculo de puntos de función.
10. Enuncie medidas de calidad de un proyecto software.
11. Defina eficacia de eliminación de defectos.
12. Fundamente el establecimiento de una línea base de métricas.
13. Caracterice los atributos deseables de una línea base.
14. De un esquema de proceso de recopilación de métricas del software.
15. Enuncie los tres componentes de un programa de medida exacto propuesto por Basili.
16. Enuncie las tres etapas del paradigma de mejora de calidad asociado al método OPM (Objetivo/Pregunta/Métrica).
17. Enuncie los pasos para el establecimiento de un programa de métricas de software.

PROCESO DE SOFTWARE Y MÉTRICAS DE PROYECTOS

La medición es fundamental para cualquier disciplina de ingeniería, y la ingeniería del software no es una excepción. La medición nos permite tener una visión más profunda proporcionando un mecanismo para la evaluación objetiva. Lord Kelvin en una ocasión dijo:

Cuando pueda medir lo que está diciendo y expresarlo con números, ya conoce algo sobre ello; cuando no pueda medir, cuando no pueda expresar lo que dice con números, su conocimiento es precario y deficiente: puede ser el comienzo del conocimiento, pero en sus pensamientos, apenas está avanzando hacia el escenario de la ciencia.

La comunidad de la ingeniería del software ha comenzado finalmente a tomarse en serio las palabras de Lord Kelvin. Pero no sin frustraciones y sí con gran controversia.

Las *métricas del software* se refieren a un amplio elenco de mediciones para el software de computadora. La medición se puede aplicar al proceso del software con el intento de mejorarlo sobre una base continua. Se puede utilizar en el proyecto del software para ayudar en la estimación, el control de calidad, la evaluación de productividad y el control de proyectos. Finalmente, el ingeniero de software puede utilizar la medición para ayudar a evaluar la calidad de los resultados de trabajos técnicos y para ayudar en la toma de decisiones táctica a medida que el proyecto evoluciona.

VISTAZO RÁPIDO

¿Qué es? El proceso del software y las métricas del producto son una medida cuantitativa que permite a la gente del software tener una visión profunda de la eficacia del proceso del software y de los proyectos que dirigen utilizando el proceso como un marco de trabajo. Se reúnen los datos básicos de calidad y productividad. Estos datos son entonces analizados, comparados con promedios anteriores, y evaluados para determinar las mejoras en la calidad y productividad. Las métricas son también utilizadas para señalar áreas con problemas de manera que se puedan desarrollar los remedios y mejorar el proceso del software.

¿Quién lo hace? Las métricas del software son analizadas y evaluadas por los administradores del software. A menudo las medidas son reunidas por los ingenieros del software.

¿Por qué es importante? Si no mides, sólo podrás juzgar basándote en una evaluación subjetiva. Mediante la medición, se pueden señalar las tendencias (buenas o malas), realizar mejores estimaciones, llevar a cabo una verdadera mejora sobre el tiempo.

¿Cuáles son los pasos? Comenzar definiendo un conjunto limitado de medidas de procesos, proyectos y productos que sean fáciles de recoger. Estas medidas son a menudo normalizadas utili-

zando métricas orientadas al tamaño o a la función. El resultado se analiza y se compara con promedios anteriores de proyectos similares realizados con la organización. Se evalúan las tendencias y se generan las conclusiones.

¿Cuál es el producto obtenido? Es un conjunto de métricas del software que proporcionan una visión profunda del proceso y de la comprensión del proyecto.

¿Cómo puedo estar seguro de que lo he hecho correctamente? Aplicando un plan de medición sencillo pero consistente, **que** nunca utilizaremos para evaluar, premiar o castigar el rendimiento individual.

Dentro del contexto de la gestión de proyectos de software, en primer lugar existe una gran preocupación por las métricas de productividad y de calidad —medidas «de salida» (finalización) del desarrollo del software, basadas en el esfuerzo y tiempo empleados, y medidas de la «utilidad» del producto obtenido—.

Park, Goethert y Florac [PAR96] tratan en su guía de la medición del software las razones por las que medimos:

Hay cuatro razones para medir los procesos del software, los productos y los recursos:

- caracterizar
- evaluar
- predecir
- mejorar

Caracterizamos para comprender mejor los procesos, los productos, los recursos y los entornos y para establecer las líneas base para las comparaciones con evaluaciones futuras.

Evaluamos para determinar el estado con respecto al diseño. Las medidas utilizadas son los sensores que nos permiten conocer cuándo nuestros proyectos y nuestros procesos están perdiendo la pista, de modo que podamos ponerlos bajo control. También evaluamos para valorar la consecución de los objetivos de calidad y para evaluar el impacto de la tecnología y las mejoras del proceso en los productos y procesos.

Predecimos para poder planificar. Realizar mediciones para la predicción implica aumentar la comprensión de las relaciones entre los procesos y los productos y la construcción de modelos de estas relaciones, por lo que los valores que observamos para algunos atributos pueden ser utilizados para predecir otros. Hacemos esto porque queremos establecer objetivos alcanzables para el coste, planificación, y calidad — de manera que se puedan aplicar los recursos apropiados—. Las medidas de predicción también son la base para la extrapolación de tendencias, con lo que la estimación para el coste, tiempo y calidad se puede actualizar basándose en la evidencia actual. Las proyecciones y las estimaciones basadas en datos históricos también nos ayudan a analizar riesgos y a realizar intercambios diseño/coste.

Medimos para mejorar cuando recogemos la información cuantitativa que nos ayuda a identificar obstáculos, problemas de raíz, ineficiencias y otras oportunidades para mejorar la calidad del producto y el rendimiento del proceso.

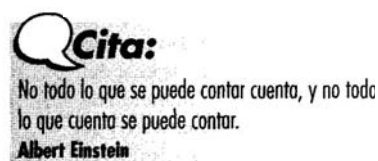
4.1 MEDIDAS, MÉTRICAS E INDICADORES

Aunque los términos *medida*, *medición* y *métricas* se utilizan a menudo indistintamente, es importante destacar las diferencias sutiles entre ellos. Como los términos «medida» y «medición» se pueden utilizar como un nombre o como un verbo, las definiciones de estos términos se pueden confundir. Dentro del contexto de la ingeniería del software, una *medida* proporciona una indicación cuantitativa de la extensión, cantidad, dimensiones, capacidad o tamaño de algunos atributos de un proceso o producto. La *medición* es el acto de determinar una medida. El *IEEE Standard Glossary of Software Engineering Terms* [IEEE93] define *métrica* como «una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado».

Cuando, simplemente, se ha recopilado un solo aspecto de los datos (por ejemplo: el número de errores descubiertos en la revisión de un módulo), se ha establecido una medida. La medición aparece como resultado de la recopilación de uno o varios aspectos de los datos (por ejemplo: se investiga un número de revisiones de módulos para recopilar medidas del número de errores encontrados durante cada revisión). Una métrica del software relata de alguna forma las medidas individuales sobre algún aspecto (por ejemplo: el número medio de errores encontrados por revisión o el número medio de errores encontrados por persona y hora en revisiones¹).

Un ingeniero del software recopila medidas y desarrolla métricas para obtener indicadores. Un *indicador* es una métrica o una combinación de métricas que proporcionan una visión profunda del proceso del software, del proyecto de software o del producto en sí

[RAG95]. Un indicador proporciona una visión profunda que permite al gestor de proyectos o a los ingenieros de software ajustar el producto, el proyecto o el proceso para que las cosas salgan mejor.



Por ejemplo, cuatro equipos de software están trabajando en un proyecto grande de software. Cada equipo debe conducir revisiones del diseño, pero puede seleccionar el tipo de revisión que realice. Sobre el examen de la métrica, *errores encontrados por persona-hora consumidas*, el gestor del proyecto notifica que dos equipos que utilizan métodos de revisión más formales presentan un 40 por 100 más de *errores encontrados por persona-hora consumidas* que otros equipos. Suponiendo que todos los parámetros son iguales, esto proporciona al gestor del proyecto un indicador en el que los métodos de revisión más formales pueden proporcionar un ahorro mayor en inversión de tiempo que otras revisiones con un enfoque menos formal. Esto puede sugerir que todos los equipos utilicen el enfoque más formal. La métrica proporciona al gestor una visión más profunda. Y además le llevará a una toma de decisiones más fundamentada.

¹ Esto asume que se recopila otra medida, persona y horas gastadas en cada revisión.

4.2 MÉTRICAS EN EL PROCESO Y DOMINIOS DEL PROYECTO

La medición es algo común en el mundo de la ingeniería. Se mide el consumo de energía, el peso, las dimensiones físicas, la temperatura, el voltaje, la relación señal-ruido... la lista es casi interminable. Por desgracia, la medición es mucho menos común en el mundo de la ingeniería del software. Existen problemas para ponerse de acuerdo sobre qué medir y las medidas de evaluación de problemas recopilados.

Referencia Web

Se puede descargar una guía completa de métricas del software desde:

www.inv.nasa.gov/SWG/resources/NASA-GB-001-94.pdf

Se deberían recopilar métricas para que los indicadores del proceso y del producto puedan ser ciertos. Los *indicadores de proceso* 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, productos de trabajo e hitos). También permiten que los gestores evalúen lo que funciona y lo que no. Las métricas del proceso se recopilan de todos los proyectos y durante un largo período de tiempo. Su intento es proporcionar indicadores que lleven a mejoras de los procesos de software a largo plazo.

Los *indicadores de proyecto* permiten al gestor de proyectos del software (1) evaluar el estado del proyecto en curso; (2) seguir la pista de los riesgos potenciales; (3) detectar las áreas de problemas antes de que se conviertan en «críticas»; (4) ajustar el flujo y las tareas del trabajo, y (5) evaluar la habilidad del equipo del proyecto en controlar la calidad de los productos de trabajo del software.

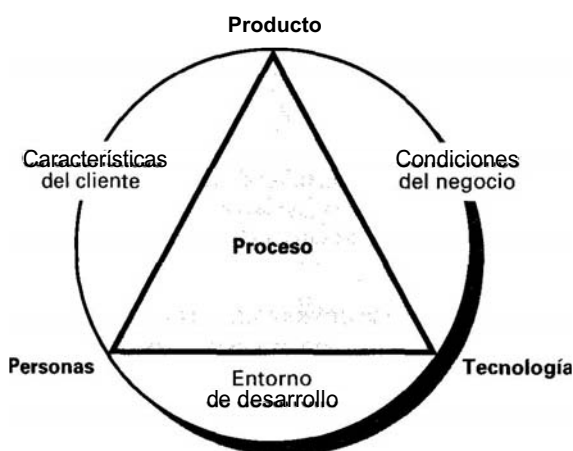


FIGURA 4.1. Determinantes de la calidad del software y de la efectividad de organización (adaptado según [PAU94]).

En algunos casos, se pueden utilizar las mismas métricas del software para determinar tanto el proyecto como los indicadores del proceso. En realidad, las medidas que recopila un equipo de proyecto y las convierte en métricas para utilizarse durante un proyecto también pueden transmitirse a los que tienen la responsabilidad de mejorar el proceso del software. Por esta razón, se utilizan muchas de las mismas métricas tanto en el dominio del proceso como en el del proyecto.

4.2.1. Métricas del proceso y mejoras en el proceso del software

La Única forma racional de mejorar cualquier proceso es medir atributos del proceso, desarrollar un juego de métricas significativas según estos atributos y entonces utilizar las métricas para proporcionar indicadores que conducirán a una estrategia de mejora. Pero antes de estudiar las métricas del software y su impacto en la mejoras de los procesos del software es importante destacar que el proceso es el Único factor de «los controlables al mejorar la calidad del software y su rendimiento como organización» [PAU94].

CLAVE

La habilidad y la motivación del personal realizando el trabajo son los factores más importantes que influyen en la calidad del software.

En la Figura 4.1, el proceso se sitúa en el centro de un triángulo que conecta tres factores con una profunda influencia en la calidad del software y en el rendimiento como organización. La destreza y la motivación del personal se muestran como el Único factor realmente influyente en calidad y en el rendimiento [BOE81]. La complejidad del producto puede tener un impacto sustancial sobre la calidad y el rendimiento del equipo. La tecnología (por ejemplo: métodos de la ingeniería del software) que utiliza el proceso también tiene su impacto. Además, el triángulo de proceso existe dentro de un círculo de condiciones de entornos que incluyen entornos de desarrollo (por ejemplo: herramientas CASE), condiciones de gestión (por ejemplo: fechas tope, reglas de empresa) y características del cliente (por ejemplo: facilidad de comunicación).



¿Cómo puedo medir la efectividad de un proceso de software?

La eficacia de un proceso de software se mide indirectamente. Esto es, se extrae un juego de métricas según los resultados que provienen del proceso. Entre los resultados se incluyen medidas de errores detectados antes de la entrega del software, defectos detectados e informados a los usuarios finales, productos de trabajo entregados (productividad), esfuerzo humano y tiempo consumido, ajuste con la planificación y otras medidas. Las métricas de proceso también se extraen midiendo las características de tareas específicas de la ingeniería del software. Por ejemplo, se podría medir el tiempo y el esfuerzo de llevar a cabo las actividades de protección y las actividades genéricas de ingeniería del software del Capítulo 2.

Grady [GRA92] argumenta que existen unos usos «privados y públicos» para diferentes tipos de datos de proceso. Como es natural que los ingenieros del software pudieran sentirse sensibles ante la utilización de métricas recopiladas sobre una base particular, estos datos deberían ser *privados* para el individuo y servir sólo como un indicador de ese individuo. Entre los ejemplos de *métricas privadas* se incluyen: índices de defectos (individualmente), índices de defectos (por módulo), errores encontrados durante el desarrollo.

La filosofía de «datos de proceso privados» se ajusta bien con el enfoque del *proceso personal del software* propuesto por Humphrey [HUM95]. Humphrey describe el enfoque de la manera siguiente:

El proceso personal del software (PPS) es un conjunto estructurado de descripciones de proceso, mediciones y métodos que pueden ayudar a que **los** ingenieros mejoren **su** rendimiento personal. Proporcionan las formas, guiones y estándares que les ayudan a estimar y planificar **su** trabajo. Muestra cómo definir procesos y cómo medir **su** calidad y productividad. Un principio PPS fundamental es que todo el mundo es diferente y que un método que sea efectivo para un ingeniero puede que no sea adecuado para otro. Así pues, el PPS ayuda a que **los** ingenieros midan y sigan la pista de su trabajo para que puedan encontrar **los** métodos que sean mejores para ellos.

Humphrey reconoce que la mejora del proceso del software puede y debe empezar en el nivel individual. Los datos privados de proceso pueden servir como referencia importante para mejorar el trabajo individual del ingeniero del software.

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 función². El equipo revisa estos datos para detectar los indicadores que pueden mejorar el rendimiento del equipo.

² Consulte las Secciones 4.3.1 y 4.3.2 para estudios más detallados sobre LDC (líneas de código) y métricas de puntos de función.



Las métricas públicas permiten a una organización realizar cambios estratégicos que mejoran el proceso del software y cambios tácticos durante un proyecto de software.

Las métricas públicas generalmente asimilan información que originalmente era privada de particulares y equipos. Los índices de defectos a nivel de proyecto (no atribuidos absolutamente a un particular), esfuerzo, tiempo y datos afines se recopilan y se evalúan en un intento de detectar indicadores que puedan mejorar el rendimiento del proceso organizativo.

Las métricas del proceso del software pueden proporcionar beneficios significativos a medida que una organización trabaja por mejorar su nivel global de madurez del proceso. Sin embargo, al igual que todas las métricas, éstas pueden usarse mal, originando más problemas de los que pueden solucionar. Grady [GRA92] sugiere una «etiqueta de métricas del software» adecuada para gestores al tiempo que instituyen un programa de métricas de proceso:

- Utilice el sentido común y una sensibilidad organizativa al interpretar datos de métricas.
- Proporcione una retroalimentación regular para 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 de proceso.
- No se obsesione con una sola métrica y excluya otras métricas importantes.



¿Qué directrices deben aplicarse cuando recogemos métricas del software?

A medida que una organización está más a gusto con la recopilación y utiliza métricas de proceso, la derivación de indicadores simples abre el camino hacia un enfoque más riguroso llamado *mejora estadística de proceso del software* (MEPS). En esencia, MEPS utiliza el análisis de fallos del software para recopilar infor-

mación de errores y defectos³ encontrados al desarrollar y utilizar una aplicación de sistema o producto. El análisis de fallos funciona de la misma manera:

Referencia Web

MPSE y otra información relacionada con la calidad está disponible en la Sociedad Americana para la Calidad en www.asq.org

1. Todos los errores y defectos se categorizan por origen (por ejemplo: defectos en la especificación, en la lógica, no acorde con los estándares).
2. Se registra tanto el coste de corregir cada error como el del defecto.
3. El número de errores y de defectos de cada categoría se cuentan y se ordenan en orden descendente.
4. Se computa el coste global de errores y defectos de cada categoría.
5. Los datos resultantes se analizan para detectar las categorías que producen el coste más alto para la organización.
6. Se desarrollan planes para modificar el proceso con el intento de eliminar (o reducir la frecuencia de apariciones de) la clase de errores y defectos que sean más costosos.

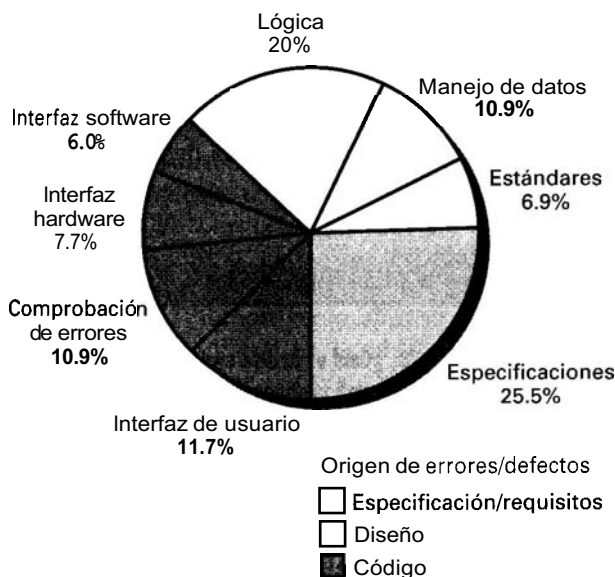


FIGURA 4.2. Causas de defectos y su origen para cuatro proyectos de software [GRA94].

³ Como se trata en el Capítulo 8, un *error* es alguna fisura en un producto de trabajo de ingeniería del software o en la entrega descubierta por los ingenieros del software antes de que el software sea entregado al usuario final.

Un defecto es una fisura descubierta después de la entrega al usuario final.



No puedes mejorar tu enfoque para la ingeniería del software a menos que comprendas donde estás fuerte y donde estás débil. Utilice las técnicas MEPS para aumentar esa comprensión.

Siguiendo los pasos 1 y 2 anteriores, se puede desarrollar una simple distribución de defectos (Fig. 4.2) [GRA94]. Para el diagrama de tarta señalado en la figura, se muestran ocho causas de defectos y sus orígenes (indicados en sombreado). Grady sugiere el desarrollo de un *diagrama de espina* [GRA92] para ayudar a diagnosticar los datos presentados en el diagrama de frecuencias. En la Figura 4.3 la espina del diagrama (la línea central) representa el factor de calidad en consideración (en este caso, los *defectos de especificación* que cuentan con el 25 por 100 del total). Cada una de las varillas (líneas diagonales) conectadas a la espina central indican una causa potencial del problema de calidad (por ejemplo: requisitos perdidos, especificación ambigua, requisitos incorrectos y requisitos cambiados). La notación de la espina y de las varillas se añade entonces a cada una de las varillas principales del diagrama para centrarse sobre la causa destacada. La expansión se muestra sólo para la causa «incorrecta» en la Figura 4.3.

La colección de métricas del proceso es el conductor para la creación del diagrama en espina. Un diagrama en espina completo se puede analizar para extraer los indicadores que permitan a una organización de software modificar su proceso para reducir la frecuencia de errores y defectos.

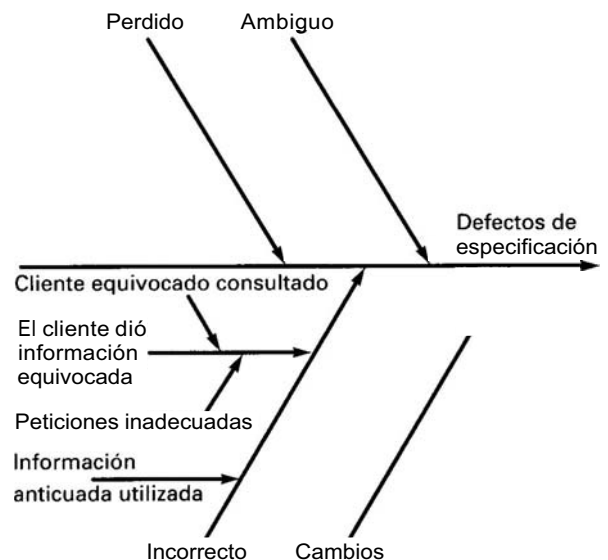


FIGURA 4.3. Un diagrama de espina (Adaptado de [GRA92]).

4.2.2. Métricas del proyecto

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

Referencia cruzada

las técnicas de estimación de proyectos se estudian en el capítulo 5.

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 del software. A medida que avanza un proyecto, las medidas del esfuerzo y del tiempo consumido se comparan con las estimaciones originales (y la planificación de proyectos). El gestor de proyectos utiliza estos datos para supervisar y controlar el avance.

A medida que comienza el trabajo técnico, otras métricas de proyectos 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 (Capítulos 19 al 24) para evaluar la calidad del diseño y para proporcionar indicadores que influirán en el enfoque tomado para la generación y prueba del código.

La utilización de métricas para el proyecto tiene dos aspectos fundamentales. En primer lugar, estas métricas se utilizan para minimizar la planificación de desarrollo haciendo los ajustes necesarios que eviten retrasos y reduzcan problemas y riesgos 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, modificando el enfoque técnico que mejore la calidad.



¿Cómo debemos utilizar las métricas durante el proyecto?

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

Otro modelo de métricas del proyecto de software [HET93] sugiere que todos los proyectos deberían medir:

- entradas: la dimensión de los recursos (p. ej.: personas, entorno) que se requieren para realizar el trabajo,
- salidas: medidas de las entregas o productos creados durante el proceso de ingeniería del software,
- resultados: 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 de trabajo cuando fluyen de una actividad (o tarea) a la siguiente.

4.3 MEDICIONES DEL SOFTWARE

Las mediciones del mundo físico se pueden categorizar de dos maneras; *medidas directas* (por ejemplo: la longitud de un tomillo) y *medidas indirectas* (por ejemplo: la «calidad» de los tomillos producidos, medidos contando los artículos defectuosos). Las métricas del software se pueden categorizar de forma similar.

Entre las medidas directas del proceso de la ingeniería del software se incluyen el coste y el esfuerzo aplicados. Entre las medidas directas del producto se incluyen las líneas de código (LDC) producidas, velocidad de ejecución, tamaño de memoria, y los defectos informados durante un período de tiempo establecido. Entre las medidas indirectas se incluyen la funcionalidad, calidad, complejidad, eficiencia, fiabilidad, facilidad de mantenimiento y muchas otras «capacidades» tratadas en el Capítulo 19.



¿Cuál es la diferencia entre medidas directas e indirectas?

El coste y el esfuerzo requerido para construir el software, el número de líneas de código producidas, y otras medidas directas son relativamente fáciles de reunir, mientras que los convenios específicos para la medición se establecen más adelante. Sin embargo, la calidad y funcionalidad del software, o su eficiencia o mantenimiento son más difíciles de evaluar y sólo pueden ser medidas indirectamente.

El dominio de las métricas del software se dividen en métricas de proceso, proyecto y producto. También se acaba de destacar que las métricas de producto que

son privadas para un individuo a menudo se combinan para desarrollar métricas del proyecto que sean públicas para un equipo de software. Las métricas del proyecto se consolidan para crear métricas de proceso que sean públicas para toda la organización del software. Pero ¿cómo combina una organización métricas que provengan de particulares o proyectos?



Puesto que muchos factores influyen en el trabajo del software, no utilice las métricas para comparar personas o equipos.

Para ilustrarlo, se va a tener en consideración un ejemplo sencillo. Personas de dos equipos de proyecto diferentes registran y categorizan todos los errores que se encuentran durante el proceso del software. Las medidas de las personas se combinan entonces para desarrollar medidas de equipo. El equipo A encontró 342 errores durante el proceso del software antes de su realización. El equipo B encontró 184. Considerando que en el resto los equipos son iguales, ¿qué equipo es más efectivo en detectar errores durante el proceso? Como no se conoce el tamaño o la complejidad de los proyectos, no se puede responder a esta pregunta. Sin embargo, si se normalizan las medidas, es posible crear métricas de software que permitan comparar medidas más amplias de otras organizaciones.

4.3.1. 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 4.4. La tabla lista cada proyecto de desarrollo de software de los últimos años y las medidas correspondientes de cada proyecto. Refiriéndonos a la entrada de la tabla (Fig. 4.4) del proyecto *alfa*: se desarrollaron 12.100 líneas de código (LDC) con 24 personas-mes y con un coste de £168.000. Debe tenerse en cuenta que el esfuerzo y el coste 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 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.



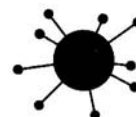
¿Qué datos deberíamos reunir para derivar métricas orientadas al tamaño?

Proyecto	LDC	Esfuerzo	Coste £(000)	Pag. 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 4.4. Métricas orientadas al tamaño.

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

- errores por KLDC (miles de líneas de código)
- defectos⁴ por KLDC
- E por LDC
- páginas de documentación por KLDC



Plantilla de colección de métricas

Además, se pueden calcular otras métricas interesantes:

- errores por persona-mes
- LDC por persona-mes
- £ por página de documentación

4.3.2. Métricas Orientadas a la Función

Las métricas del software orientadas a la función utilizan una medida de la funcionalidad entregada por la aplicación como un valor de normalización. Ya que la «funcionalidad» no se puede medir directamente, se debe derivar indirectamente mediante otras medidas directas. Las métricas orientadas a la función fueron propuestas por primera vez por Albretch [ALB79], quien sugirió una medida llamada *punto defunción*. Los puntos de función se derivan con una relación empírica

⁴ Un defecto ocurre cuando las actividades de garantía de calidad (p. ej.: revisiones técnicas formales) fallan para descubrir un error en un producto de trabajo generado durante el proceso del software.

según las medidas contables (directas) del dominio de información del software y las evaluaciones de la complejidad del software.



Se puede obtener información completo sobre los puntos de función en: www.ifpug.org

Los puntos de función se calculan [IFP94] completando la tabla de la Figura 4.5. Se determinan cinco características de dominios de información y se proporcionan las cuentas en la posición apropiada de la tabla. Los valores de los dominios de información se definen de la forma siguiente?

Número de entradas de usuario. Se cuenta cada entrada de usuario que proporciona diferentes datos orientados a la aplicación. Las entradas se deberían diferenciar de las peticiones, las cuales se cuentan de forma separada.



Los puntos de función se derivan de medidas directas del dominio de la información.

Número de salidas de usuario. Se cuenta cada salida que proporciona al usuario información orientada a la aplicación. En este contexto la salida se refiere a informes, pantallas, mensajes de error, etc. Los elementos de datos particulares dentro de un informe no se cuentan de forma separada.

Número de peticiones de usuario. Una petición se define como una entrada interactiva que produce la generación de alguna respuesta del software inmediata en forma de salida interactiva. Se cuenta cada petición por separado.

Número de archivos. Se cuenta cada archivo maestro lógico (esto es, un grupo lógico de datos que puede ser una parte de una gran base de datos o un archivo independiente).

Número de interfaces externas. Se cuentan todas las interfaces legibles por la máquina (por ejemplo: archivos de datos de cinta o disco) que se utilizan para transmitir información a otro sistema.

Una vez que se han recopilado los datos anteriores, a la cuenta se asocia un valor de complejidad. Las organizaciones que utilizan métodos de puntos de función desarrollan criterios para determinar si una entrada en particular es simple, media o compleja. No obstante la determinación de la complejidad es algo subjetiva.

Parámetros de medición	Factor de ponderación				
	Cuenta	Simple	Medio	Complejo	
Número de entradas de usuario	<input type="text"/>	x 3	4	6	= <input type="text"/>
Número de salidas de usuario	<input type="text"/>	x 4	5	7	= <input type="text"/>
Número de peticiones de usuario	<input type="text"/>	x 3	4	6	= <input type="text"/>
Número de archivos	<input type="text"/>	x 7	10	15	= <input type="text"/>
Número de interfaces externas	<input type="text"/>	x 5	7	10	= <input type="text"/>
Cuenta total	→				<input type="text"/>

FIGURA 4.5. Cálculo de puntos de función.

Para calcular puntos de función (PF), se utiliza la relación siguiente:

$$PF = \text{cuenta-total} \times [0,65 + 0,01 \times 6(F_i)] \quad (4.1)$$

en donde cuenta-total es la suma de todas las entradas PF obtenidas de la figura 4.5.

F_i ($i = 1$ a 14) son «valores de ajuste de la complejidad» según las respuestas a las siguientes preguntas [ART85]:

1. ¿Requiere el sistema copias de seguridad y de recuperación fiables?
2. ¿Se requiere comunicación de datos?
3. ¿Existen funciones de procesamiento distribuido?
4. ¿Es crítico el rendimiento?
5. ¿Se ejecutará el sistema en un entorno operativo existente y fuertemente utilizado?
6. ¿Requiere el sistema entrada de datos interactiva?
7. ¿Requiere la entrada de datos interactiva que las transacciones de entrada se lleven a cabo sobre múltiples pantallas u operaciones?
8. ¿Se actualizan los archivos maestros de forma interactiva?
9. ¿Son complejas las entradas, las salidas, los archivos o las peticiones?
10. ¿Es complejo el procesamiento interno?
11. ¿Se ha diseñado el código para ser reutilizable?
12. ¿Están incluidas en el diseño la conversión y la instalación?
13. ¿Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones?
14. ¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?

⁵ En realidad, la definición de los valores del dominio de la información y la forma en que se cuentan es un poco más complejo. El lector interesado debería consultar [IFP94] para obtener más detalles.

Cada una de las preguntas anteriores es respondida usando una escala con rangos desde 0 (no importante o aplicable) hasta 5 (absolutamente esencial). Los valores constantes de la ecuación (4.1) y los factores de peso que se aplican a las cuentas de los dominios de información se determinan empíricamente.

Una vez que se han calculado los puntos de función, se utilizan de forma análoga a las LDC como forma de normalizar las medidas de productividad, calidad y otros atributos del software.

4.3.3. Métricas ampliadas de punto de función

La medida de punto de función se diseñó originalmente para aplicarse a aplicaciones de sistemas de información de gestión. Para acomodar estas aplicaciones, se enfatizó la dimensión de datos (los valores de dominios de información tratados anteriormente) para la exclusión de dimensiones (control) funcionales y de comportamiento. Por esta razón, la medida del punto de función era inadecuada para muchos sistemas de ingeniería y sistemas empotrados (que enfatizan función y control). Para remediar esta situación se ha propuesto un número de extensiones a la métrica del punto de función básica.

PUNTO CLAVE

La extensión de los puntos de función se utiliza en la ingeniería, en las aplicaciones de tiempo real y en las aplicaciones orientadas al control.

Una extensión del punto de función es la llamada puntos de características [JON91]; es una ampliación de la medida del punto de función que se puede aplicar a sistemas y aplicaciones de ingeniería del software. La medida de punto de característica acomoda a aplicaciones en donde la complejidad del algoritmo es alta. Las aplicaciones de software de tiempo real, de control de procesos, y empotradas tienden a tener alta complejidad de algoritmos y por lo tanto son adecuadas para el punto de característica.

Para calcular el punto de característica, los valores de dominio de información se cuentan otra vez y se pesan de la forma que se describe en la Sección 4.3.2. Además, la métrica del punto de característica cuenta una característica nueva del software—los *algoritmos*—. Un algoritmo se define como «un problema de cálculo limitado que se incluye dentro de un programa de computadora específico» [JON91]. Invertir una matriz, decodificar una cadena de bits o manejar una interrupción son ejemplos de algoritmos.

⁶ Debe señalarse que también han sido propuestas otras extensiones a los puntos de función para aplicarlas en trabajos de software de tiempo real (p.ej., [ALA97]). Sin embargo, ninguno de estos parece usarse ampliamente en la industria.

Referencia Web

Se puede encontrar una lista útil de preguntas realizadas con frecuencia sobre los puntos de función (y puntos de función extendidos) en:

<http://ourworld.compuserve.com/homepages/softcomp/>

Boeing ha desarrollado otra extensión de punto de función para sistemas en tiempo real y productos de ingeniería. El enfoque de Boeing integra la dimensión de datos del software con las dimensiones funcionales y de control para proporcionar una medida orientada a la función que es adecuada a aplicaciones que enfatizan las capacidades de control y función. Las características de las tres dimensiones del software se «cuentan, cuantifican y transforman» en una medida que proporciona una indicación de la funcionalidad entregada por el software⁶, llamada *Punto de Función 3D* [WHI95]

La *dimensión de datos* se evalúa exactamente igual a como se describe en la Sección 4.3.2. Las cuentas de datos retenidos (la estructura interna de datos de programas, p. ej.: archivos) y los datos externos (entradas, salidas, peticiones y referencias externas) se utilizan a lo largo de las medidas de la complejidad para derivar una cuenta de dimensión de datos.

La *dimensión funcional* se mide considerando «el número de operaciones internas requeridas para transformar datos de entrada en datos de salida» [WHI95]. Para propósitos de computación de los puntos de función 3D, se observa una «transformación» como una serie de pasos de proceso que quedan limitados por sentencias semánticas. La *dimensión de control* se mide contando el número de transiciones entre estados⁷.

Un estado representa algún modo de comportamiento externamente observable, y una transición ocurre como resultado de algún suceso que cambia el modo de comportamiento del sistema o del software (esto es, cambia el estado). Por ejemplo, un teléfono inalámbrico contiene software que soporta funciones de marcado automático. Para introducir el estado de marcado automático desde un estado en descanso, el usuario pulsa una tecla **Auto** en el teclado numérico. Este suceso hace que una pantalla LCD pida un código que indique la parte que se llama. Con la entrada del código y pulsando la **tecla de Marcado** (otro suceso), el software del teléfono inalámbrico hace una transición al estado de marcado. Cuando se computan puntos de función 3D, no se asigna un valor de complejidad a las transiciones.

⁷ En el capítulo 12 se presenta un estudio detallado de la dimensión de comportamientos que incluyen estados y transiciones de estados.

Para calcular puntos de función 3D, se utiliza la relación siguiente:

$$\text{índice} = I + O + Q + F + E + T + R \quad (4.2)$$

en donde I, O, Q, F, E, T y R representan valores con peso de complejidad en los elementos tratados anteriormente: entradas, salidas, peticiones, estructuras de datos internas, archivos externos, transformaciones y transiciones, respectivamente. Cada valor con peso de complejidad se calcula con la relación siguiente:

valor con peso de complejidad =

$$= N_{il} W_{il} + N_{ia} W_{ia} + N_{ih} W_{ih} \quad (4.3)$$

en donde N_{il} , N_{ia} y N_{ih} representan el número de apariciones del elemento i (p. ej.: salidas) para cada nivel de complejidad (bajo, medio, alto), y W_{il} , W_{ia} y W_{ih} son los pesos correspondientes. El cálculo global de los puntos de función 3D se muestran en la Figura 4.6.

Se debería señalar que los puntos de función, los puntos de característica y los puntos de función 3D representan lo mismo —«funcionalidad» o «utilidad» entregada por el software—. En realidad, cada una de estas medidas producen el mismo valor si sólo se considera la dimensión de datos de una aplicación. Para sistemas de tiempo real más complejos, la cuenta de puntos de característica a menudo se encuentra entre el 20 y el 35 por 100 más que la cuenta determinada sólo con los puntos de función.

El punto de función (y sus extensiones), como la medida LDC, es controvertido. Los partidarios afirman que PF es independiente del lenguaje de programación, lo cual es ideal para aplicaciones que utilizan lenguajes convencionales y no procedimentales; esto se basa en los datos que probablemente se conocen al principio de la evolución de un proyecto, y así el PF es más atractivo como enfoque de estimación.

Pasos de proceso \ Sentencias semánticas	Sentencias semánticas		
	1-5	6-10	11+
1-10	Bajo	Bajo	Medio
11-20	Bajo	Medio	Alto
21+	Medio	Alto	Alto

FIGURA 4.6. Cálculo de los índices de los puntos de función 3D.

Los detractores afirman que el método requiere algún «juego de manos» en el que el cálculo se base en datos subjetivos y no objetivos; y afirman que las cuentas del dominio de información (y otras dimensiones) pueden ser difíciles de recopilar después de realizado, y por último que el PF no tiene un significado físico directo —es simplemente un número—.


4.4 RECONCILIACIÓN DE LOS 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. Ciertos estudios han intentado relacionar las métricas LDC y PF. Citando a Albrecht y Gaffney [ALB83]:

La tesis de este trabajo es que la cantidad de función proporcionada por la aplicación (programa) puede ser estimada por la descomposición de los principales componentes⁸ de datos que usa o proporciona el programa. Además, esta estimación de la función debe ser relacionada con el total de LDC a desarrollar y con el esfuerzo de desarrollo necesario.

La tabla siguiente [JON98] proporciona estimaciones informales del número medio de líneas de código que se requiere para construir un punto de función en varios lenguajes de programación:

Lenguaje de programación	LDC/PF (media)
Ensamblador	320
C	128
COBOL	106
FORTTRAN	106
Pascal	90
C++	64
Ada95	53
Visual Basic	32
Smalltalk	22
Powerbuilder (generador de código)	16
SQL	12

 Si conoces el número de LDC's, ¿es posible estimar el número de puntos de función?

⁸ Es importante destacar que la «la individualización de componentes importantes» se puede interpretar de muchas maneras. Algunos ingenieros del software que trabajan en un entorno de desarrollo orientado a objetos (Cuarta Parte) utilizan ciertas clases u objetos como métrica dominante del tamaño. Una organi-



Utilice el repaso de los datos de un modo juicioso.
Es bastante mejor calcular los PF utilizando los métodos utilizados anteriormente.

zación de mantenimiento podría observar el tamaño de los proyectos en relación con los pedidos de cambio de ingeniería (Capítulo 9). Una organización de sistemas de información podría observar el número de procesos de negocio a los que impacta una aplicación.

Una revisión de los datos anteriores indica que una LDC de C++ proporciona aproximadamente 1,6 veces la «funcionalidad» (de media) que una LDC de FORTRAN. Además, una LDC de Visual Basic proporciona 3 veces la funcionalidad de una LDC de un lenguaje de programación convencional. En [JON98] se presentan datos más precisos sobre las relaciones entre los PF y las LDC, que pueden ser usados para «repasar» programas existentes, determinando sus medidas en PF (por ejemplo, para calcular el número de puntos de función cuando se conoce la cantidad de LDC entregada).

Las medidas LDC y PF se utilizan a menudo para extraer métricas de productividad. Esta invariabilidad conduce al debate sobre el uso de tales datos. Se debe

comparar la relación LDC/personas-mes (ó PF/personas-mes) de un grupo con los datos similares de otro grupo? ¿Deben los gestores evaluar el rendimiento de las personas usando estas métricas? La respuesta a estas preguntas es un rotundo «No». La razón para esta respuesta es que hay muchos factores que influyen en la productividad, haciendo que la comparación de «manzanas y naranjas» sea mal interpretada con facilidad.

Se han encontrado los puntos de función y las LDC basadas en métricas para ser predicciones relativamente exactas del esfuerzo y coste del desarrollo del software. Sin embargo, para utilizar LDC y PF en las técnicas de estimación (capítulo 5), debe establecerse una línea base de información histórica.

4.5 MÉTRICAS PARA LA CALIDAD DEL SOFTWARE

El objetivo primordial de la ingeniería del software es producir un sistema, aplicación o producto de alta calidad. Para lograr este objetivo, los ingenieros del software deben aplicar métodos efectivos junto con herramientas modernas dentro del contexto de un proceso maduro de desarrollo de software. Además, un buen ingeniero del software (y buenos gestores de la ingeniería del software) deben medir si la alta calidad se va a llevar a cabo.

La calidad de un sistema, aplicación o producto es tan bueno como los requisitos que describen el problema, el diseño que modela la solución, el código que conduce a un programa ejecutable, y las pruebas que ejercitan el software para detectar errores. Un buen ingeniero del software utiliza mediciones que evalúan la calidad del análisis y los modelos de diseño, el código fuente, y los casos de prueba que se han creado al aplicar la ingeniería del software. Para lograr esta evaluación de la calidad en tiempo real, el ingeniero debe utilizar *medidas técnicas* (Capítulos 19 y 24) que evalúan la calidad con objetividad, no con subjetividad.



Referencia Web

Se puede encontrar una excelente fuente de información sobre la calidad del software y temas relacionados (incluyendo métricas) en: www.qualityworld.com

El gestor de proyectos también debe evaluar la calidad objetivamente, y no subjetivamente. A medida que el proyecto progresa el gestor del proyecto también debe evaluar la calidad. Las métricas privadas recopiladas por ingenieros del software particulares se asimilan para proporcionar resultados en los proyectos. Aunque se pueden recopilar muchas medidas de calidad, el primer objetivo en el proyecto es medir errores y defectos. Las métricas que provienen de estas medidas proporcionan una indicación de la efectividad de las actividades de control y de la garantía de calidad en grupos o en particulares.

Referencia cruzada

En el capítulo 8 se presenta un estudio detallado sobre las actividades de garantía de calidad del software.

4.5.1. Visión general de los factores que afectan a la calidad

Hace 25 años, McCall y Cavano [MCC78] definieron un juego de factores de calidad como los primeros pasos hacia el desarrollo de métricas de la calidad del software. Estos factores evalúan el software desde tres puntos de vista distintos: (1) operación del producto (utilizándolo), (2) revisión del producto (cambiándolo), y (3) transición del producto (modificándolo para que funcione en un entorno diferente, por ejemplo, «portándolo»). Los autores, en su trabajo, describen la relación entre estos factores de calidad (lo que llaman un «marco de trabajo») y otros aspectos del proceso de ingeniería del software:

En primer lugar, el marco de trabajo proporciona un mecanismo para que el gestor del proyecto identifique lo que considera importante. Estas cualidades son atributos del software, además de su corrección y rendimiento funcional, que tiene implicaciones en el ciclo de vida. En otros factores, como son facilidad de mantenimiento y portabilidad, se ha demostrado que tienen un impacto significativo en el coste del ciclo de vida...

En segundo lugar, el marco de trabajo proporciona un medio de evaluar cuantitativamente lo bien que va progresando el desarrollo en relación con los objetivos de calidad establecidos...

En tercer lugar, el marco de trabajo proporciona más interacción del personal de QA (garantía de calidad) en el esfuerzo de desarrollo...

Por último, ... el personal de garantía de calidad puede utilizar indicaciones de calidad pobre para ayudar a identificar estándares [mejores] a enfrentar en el futuro.

Un estudio detallado del marco de trabajo de McCall y Cavano, así como otros factores de calidad, se presentan en el Capítulo 19. Es interesante destacar que casi todos los aspectos del cálculo han sufrido cambios radicales con el paso de los años desde que McCall y

Cavano hicieron su trabajo, con gran influencia, en 1978. Pero los atributos que proporcionan una indicación de la calidad del software siguen siendo los mismos.



Sorprendentemente, los factores que definían la calidad del software en el año 1970 son los mismos factores que continúan definiendo la calidad del software en la primera década de este siglo.

¿Qué significa esto? Si una organización de software adopta un juego de factores de calidad como una «lista de comprobación» para evaluar la calidad del software, es probable que el software construido hoy siga exhibiendo la calidad dentro de las primeras décadas de este siglo. Incluso, cuando las arquitecturas de cálculo sufren cambios radicales (como seguramente ocurrirá), el software que exhibe alta calidad en operación, transición y revisión continuará sirviendo también a sus usuarios.

4.5.2. Medida de la calidad

Aunque hay muchas medidas de la calidad de software, la *corrección*, *facilidad de mantenimiento*, *integridad*, y *facilidad de uso* proporcionan indicadores útiles para el equipo del proyecto. Gilb [GIL88] ha sugerido definiciones y medidas para cada uno de ellos.

Corrección. Un programa debe operar correctamente o proporcionará poco valor a sus usuarios. La corrección es el grado en el que el software lleva a cabo su función requerida. La medida más común de corrección es *defectos por KLDC*, en donde un defecto se define como una falta verificada de conformidad con los requisitos.

Facilidad de mantenimiento. El mantenimiento del software cuenta con más esfuerzo que cualquier otra actividad de ingeniería del software. La facilidad de mantenimiento es la facilidad con la que se puede corregir un programa si se encuentra un error, se puede adaptar si su entorno cambia, o mejorar si el cliente desea un cambio de requisitos. No hay forma de medir directamente la facilidad de mantenimiento; por consiguiente, se deben utilizar medidas indirectas. Una simple métrica orientada al tiempo es el *tiempo medio de cambio (TMC)*, es decir el tiempo que se tarda en analizar la petición de cambio, en diseñar una modificación adecuada, en implementar el cambio, en probarlo y en distribuir el cambio a todos los usuarios. Como media, los programas que se pueden mantener tendrán un TMC más bajo (para tipos equivalentes de cambios) que los programas que no son más fáciles de mantener.

Hitachi [TAJ81] ha utilizado una métrica orientada al coste para la capacidad de mantenimiento llamada «desperdicios» —el coste en corregir defectos encontrados después de haber distribuido el software

a sus usuarios finales—. Cuando la proporción de desperdicios en el coste global del proyecto (para muchos proyectos) se representa como una función del tiempo, el gestor puede determinar si la facilidad total del software producido por una organización de desarrollo está mejorando. Se pueden emprender acciones a partir de las conclusiones obtenidas de esa información.

Integridad. En esta época de «hackers» y «firewalls», la integridad del software ha llegado a tener mucha importancia. Este atributo mide la capacidad de un sistema para resistir ataques (tanto accidentales como intencionados) contra su seguridad. El ataque se puede realizar en cualquiera de los tres componentes del software: programas, datos y documentos.

Para medir la integridad, se tienen que definir dos atributos adicionales: amenaza y seguridad. *Amenaza* es la probabilidad (que se puede estimar o deducir de la evidencia empírica) de que un ataque de un tipo determinado ocurra en un tiempo determinado. La *seguridad* es la probabilidad (que se puede estimar o deducir de la evidencia empírica) de que se pueda repeler el ataque de un tipo determinado. La integridad del sistema se puede definir como:

$$\text{integridad} = \sum [(1 - \text{amenaza}) \times (1 - \text{seguridad})]$$

donde se suman la amenaza y la seguridad para cada tipo de ataque.

Facilidad de uso. El calificativo «amigable con el usuario» se ha convertido en omnipresente en las discusiones sobre productos de software. Si un programa no es «amigable con el usuario», frecuentemente está abocado al fracaso, incluso aunque las funciones que realice sean valiosas. La facilidad de uso es un intento de cuantificar «lo amigable que puede ser con el usuario» y se puede medir en función de cuatro características: (1) habilidad intelectual y/o física requerida para aprender el sistema; (2) el tiempo requerido para llegar a ser moderadamente eficiente en el uso del sistema; (3) aumento neto en productividad (sobre el enfoque que el sistema reemplaza) medida cuando alguien utiliza el sistema moderadamente y eficientemente; y (4) valoración subjetiva (a veces obtenida mediante un cuestionario) de la disposición de los usuarios hacia el sistema. En el Capítulo 15 se estudia más detalladamente este aspecto.

Los cuatro factores anteriores son sólo un ejemplo de todos los que se han propuesto como medidas de la calidad del software. El Capítulo 19 considera este tema con más detalle.

4.5.3. Eficacia de la Eliminación de Defectos

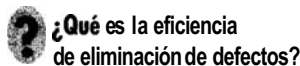
Una métrica de la calidad que proporciona beneficios tanto a nivel del proyecto como del proceso, es la *eficacia de la eliminación de defectos (EED)*. En esen-

cia, EED es una medida de la habilidad de filtrar las actividades de la garantía de calidad y de control al aplicarse a todas las actividades del marco de trabajo del proceso.

Cuando un proyecto se toma en consideración globalmente, EED se define de la forma siguiente:

$$EED = E / (E + D) \quad (4.4)$$

donde E es el número de errores encontrados antes de la entrega del software al usuario final y D es el número de defectos encontrados después de la entrega.



El valor ideal de EED es 1. Esto es, no se han encontrado defectos en el software. De forma realista, D será mayor que cero, pero el valor de EED todavía se puede aproximar a 1. Cuando E aumenta (para un valor de D dado), el valor total de EED empieza a aproximarse a 1. De hecho, a medida que E aumenta, es probable que el valor final de D disminuya (los errores se filtran antes de que se conviertan en defectos). Si se utilizan como una métrica que proporciona un indicador de la habilidad de filtrar las actividades de la garantía de la calidad y del control, EED anima a que el equipo del

proyecto de software instituya técnicas para encontrar todos los errores posibles antes de su entrega.

EED también se puede utilizar dentro del proyecto para evaluar la habilidad de un equipo en encontrar errores antes de que pasen a la siguiente actividad estructural o tarea de ingeniería del software. Por ejemplo, la tarea del análisis de los requisitos produce un modelo de análisis que se puede revisar para encontrar y corregir errores. Esos errores que no se encuentren durante la revisión del modelo de análisis se pasan a la tarea de diseño (en donde se pueden encontrar o no). Cuando se utilizan en este contexto, EED se vuelve a definir como:

$$EED_i = E_i / (E_i + E_{i+1}) \quad (4.5)$$

en donde E_i es el número de errores encontrado durante la actividad de ingeniería del software i y E_{i+1} es el número de errores encontrado durante la actividad de ingeniería del software $i + 1$ que se puede seguir para llegar a errores que no se detectaron en la actividad de la ingeniería del software i .



Utilice EED como una medida de la eficiencia de sus primeras actividades de SQA. Si EED es bajo durante el análisis y diseño, emplee algún tiempo mejorando la forma de conducir las revisiones técnicas formales.

4.8 INTEGRACIÓN DE LAS MÉTRICAS DENTRO DEL PROCESO DE INGENIERÍA DEL SOFTWARE

La mayoría de los desarrolladores de software todavía no miden, y por desgracia, la mayoría no desean ni comenzar. Como se ha señalado en este capítulo, 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 gestor de proyectos agobiado. «No entiendo por qué», se queja un profesional saturado de trabajo.

En esta sección, consideraremos algunos argumentos para las métricas del software y presentaremos un enfoque para instituir un programa de métricas dentro de una organización de ingeniería del software. Pero antes de empezar, consideremos algunas palabras de cordura sugeridas por Grady y Caswell [GRA87]:

Algunas de las cosas que describimos aquí parecen bastante fáciles. Realmente, sin embargo, establecer un programa de métricas del software con éxito es un trabajo duro. Cuando decimos esto debes esperar al menos tres años antes de que estén disponibles las tendencias organizativas, lo que puede darte alguna idea del ámbito de tal esfuerzo.

La advertencia sugerida por los autores se considera de un gran valor, pero los beneficios de la medición son tan convincentes que obligan a realizar este duro trabajo.

4.6.1. Argumentos para las Métricas del Software

¿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.



Manejamos cosas «con los números» en muchos aspectos de nuestras vidas... Estos números nos dan una visión y nos ayudan a dirigir nuestras acciones.
Michael Mah
Larry Putnam

La gestión 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. En el Capítulo 1 se señaló que el software es un aspecto de gestión estratégico para muchas compañías. Si el proceso por el que se desarrolla puede ser mejorado, se producirá un impacto directo en lo sustancial. Pero para establecer objetivos de mejora, se debe comprender el estado actual de desa-

rollo del software. Por lo tanto la medición se utiliza para establecer una línea base del proceso desde donde se pueden evaluar las mejoras.

Los rigores del trabajo diario de un proyecto del software no dejan mucho tiempo para pensar en estrategias. Los gestores de 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 medición para establecer una línea base del proyecto, cada uno de estos asuntos se hace más fácil de manejar. Ya hemos apuntado que la línea base sirve como base 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 software⁹.

Técnicamente (en el fondo), las métricas del software, cuando se aplican al producto, proporcionan 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. En posteriores capítulos examinaremos cómo se hace esto.

4.6.2. Establecimiento de una Línea Base

Estableciendo una línea base de métricas se pueden obtener beneficios a nivel de proceso, proyecto y producto (técnico). Sin embargo la información reunida no necesita ser fundamentalmente diferente. Las mismas métricas pueden servir varias veces. Las líneas base de métricas constan de datos recogidos de proyectos de software desarrollados anteriormente y pueden ser tan simples como la tabla mostrada en la figura 4.4o tan complejas como una gran base de datos que contenga docenas de medidas de proyectos y las métricas derivadas de ellos.

Para ser una ayuda efectiva en la mejora del proceso y/o en la estimación del esfuerzo y del coste, los datos de línea base deben tener los siguientes atributos: (1) los datos deben ser razonablemente exactos —se deben evitar «conjeturas» sobre proyectos pasados—; (2) los datos deben reunirse del mayor número de proyectos que sea posible;

(3) las medidas deben ser consistentes, por ejemplo, una línea de código debe interpretarse consistentemente en todos los proyectos para los que se han reunido los datos;

(4) las aplicaciones deben ser semejantes para trabajar en la estimación —tiene poco sentido utilizar una línea base obtenida en un trabajo de sistemas de información batch para estimar una aplicación empujada de tiempo real—.

4.6.3. Colección de métricas, cómputo y evaluación

El proceso que establece una línea base se muestra en la Figura 4.7. Idealmente, los datos necesarios para establecer una línea base se han ido recopilando a medida que se 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 (incuestionablemente 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 LDC y PF, así como métricas de la calidad y orientadas al proyecto. Finalmente, las métricas se deben evaluar y aplicar durante la estimación, el trabajo técnico, el control de proyectos y la mejora del proceso. 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.

PUNTO CLAVE

Los datos de las métricas de línea base deberían recogerse de una gran muestra representativa de proyectos de software del pasado.

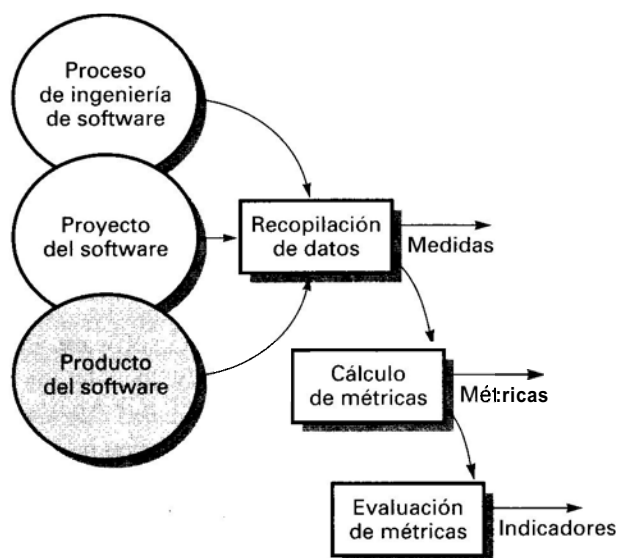


FIGURA 4.7. Proceso de recopilación de métricas del software.

⁹ Estas ideas se han formalizado en un enfoque denominado *garantía estadística de calidad del software* y se estudian en detalle en el Capítulo 8.

4.7 EL DESARROLLO DE LA MÉTRICA Y DE LA OPM (Objetivo, Pregunta, Métrica)

Uno de los problemas al que se han enfrentado los trabajadores de las métricas durante las dos últimas décadas es la de desarrollar métricas que fueran útiles para el diseñador de software. Ha sido toda una historia de utilización de la métrica dentro de los entornos industriales basada en el simple criterio de lo que era facilitar la medida, más que emplear cualquier criterio relacionado con la utilidad. El panorama de la última mitad de los años 80 y la primera mitad de la década de los 90, constató el hecho de que mientras había sido desarrollado mucho trabajo en la validación de la métrica y en el esclarecimiento de los principios teóricos detrás de ella, muy poco había sido hecho para dotar al diseñador de software con herramientas para la selección o construcción de métricas.

El objetivo de esta sección es describir OPM, que es casi con certeza el método de desarrollo de métrica más ampliamente aplicado y mejor conocido y que ha sido desarrollado por Victor Basili y sus colaboradores de la Universidad de Maryland. Basili y sus colaboradores tenían ya una larga historia de validación de métricas en la década de los 80 y el método OPM (objetivo, pregunta y métrica) surgió de un trabajo que fue desarrollado dentro de un laboratorio de ingeniería del software esponsorizado por la Agencia Americana del Espacio, NASA.

Basili establecía que para que una organización tuviera un programa de medida exacto era necesario que tuviera constancia de tres componentes:

1. Un proceso donde pudieran articularse metas u objetivos para sus proyectos.
2. Un proceso donde estas metas pudieran ser traducidas a los datos del proyecto que exactamente reflejasen dichas metas u objetivos en términos de software.
3. Un proceso que interpretara los datos del proyecto con el fin de entender los objetivos.

Un ejemplo de un objetivo típico que bien podría ser adoptado por una empresa es que la cantidad de trabajo de revisión sobre un diseño de sistema debido a problemas que fueran descubiertos por los programadores se redujera al 80 por ciento.

A partir de este ejemplo de objetivo emerge un cierto número de preguntas típicas cuya contestación podría ser necesaria para clarificar los objetivos y por consiguiente el desarrollo de una métrica. Un conjunto de ejemplos de estas preguntas, se exponen a continuación:

- ¿Cómo realizar la cuantificación de los trabajos de revisión?
- ¿Debería ser tenido en cuenta el tamaño del producto en el cálculo de la disminución de trabajos de revisión o revisiones?
- Debería ser tenido en cuenta el incremento de mano de obra de programación requerida para validaciones suplementarias y trabajos de rediseño en comparación con el proceso actual de validar un diseño corriente.

Una vez que estas cuestiones se hayan establecido, entonces es cuando puede ser desarrollada una métrica que pueda ayudar a que se cumpla el objetivo planteado que naturalmente emergerá a partir de estas cuestiones.

La importancia de OPM proviene no solamente del hecho de que es uno de los primeros intentos de desarrollar un conjunto de medidas adecuado que pueda ser aplicado al software, sino también al hecho de que está relacionado con el paradigma de mejora de procesos que ha sido discutido previamente. Basili ha desarrollado un paradigma de mejora de calidad dentro del cual el método OPM puede encajarse perfectamente. El paradigma comprende tres etapas:

- Un proceso de llevar a cabo una auditoría de un proyecto y su entorno estableciendo metas u objetivos de calidad para el proyecto y seleccionando herramientas adecuadas y métodos y tecnologías de gestión para que dichos objetivos de calidad tengan una oportunidad de cumplirse.
- Un proceso de ejecutar un proyecto y chequear los datos relacionados con esas metas u objetivos de calidad. Este proceso es llevado a cabo en conjunción con otro proceso paralelo de actuación sobre los propios datos cuando se vea que el proyecto corre peligro de no cumplir con los objetivos de calidad.
- Un proceso para el análisis de los datos del segundo paso, con el fin de poder hacer sugerencias para una mayor mejora. Este proceso implicaría el analizar los problemas ya en la etapa de recolección de datos, problemas en la implementación de las directivas de calidad y problemas en la interpretación de los datos.

Basili [BAS96] ha proporcionado una serie de plantillas que son útiles para los desarrolladores que deseen utilizar el método OPM para desarrollar métricas realistas sobre sus proyectos. Los objetivos de OPM pueden articularse por medio de tres plantillas que cubren el propósito, la perspectiva y el entorno.

La plantilla o esquema de cálculo denominada de propósito se utiliza para articular o comparar lo que está siendo analizado y el propósito de dicha parte del proyecto. Por ejemplo, un diseñador puede desear analizar la efectividad de las revisiones de diseño con el propósito de mejorar la tasa de eliminación de errores de dicho proceso o el propio diseñador puede desear analizar las normas utilizadas por su empresa con el objetivo de reducir la cantidad de mano de obra utilizada durante el mantenimiento. Una segunda plantilla está relacionada con la perspectiva. Esta plantilla pone su atención en los factores que son importantes dentro del propio proceso o producto que está siendo evaluado. Ejemplos típicos de esto incluyen los factores de calidad de la mantenibilidad, chequeo, uso y otros factores tales como el coste y la corrección. Esta plantilla se fundamenta en la perspectiva del propio proceso al que se dirige.

Hay varios enfoques que pueden hacerse sobre el proceso de desarrollo de software — el del cliente y el del diseñador son los dos más típicos — y la elección de una u otra perspectiva tiene un efecto muy grande sobre los análisis que se llevan a cabo. Por ejemplo, si un cierto proceso como una revisión de requisitos está siendo analizada con respecto al coste, la perspectiva del diseñador, es la del que desea reducir el coste del proceso en términos de hacerlo más eficiente en cuanto a la detección de errores; sin embargo, si después examinamos el mismo propósito pero desde el punto de vista del cliente, concretamente sobre si su personal puede emplear o no una gran cantidad de tiempo en dichas revisiones, entonces la perspectiva podría involucrar el plantearse un uso más eficiente de dicho tiempo empleado en las revisiones. Ambas perspectivas son válidas — a veces se solapan y a veces son antitéticas — existe, por ejemplo, una necesidad natural en el diseñador de maximizar el beneficio a la vez que el objetivo del cliente es la corrección máxima del producto y la entrega dentro del plazo. Un punto importante a recalcar es que del examen de la perspectiva del producto, el desarrollador está en una mejor posición para evaluar un proceso de software y un producto de software y también la mejora de los mismos.

Una tercera plantilla implica el entorno. Este es el contexto dentro del cual el método OPM se aplica e implica el examen del personal, la propia empresa y los entornos de recursos en los que el análisis se está llevando a cabo. Factores típicos de entorno incluyen, por ejemplo, el tipo de sistema informático que está siendo usado, las habilidades del personal implicado en el proyecto, el modelo de ciclo de vida adoptado para tal caso, la cantidad de recursos adiestrados disponibles y el área de aplicación.

Una vez que tanto el propósito como la perspectiva y el entorno de un objetivo han sido bien especificados, el proceso de planteamiento de cuestiones y el desarrollo de una métrica o valoración puede comenzar. Antes de examinar este proceso, merece la pena dar algunos ejemplos de objetivos empleados en los términos planteados. El primero se describe a continuación:

El objetivo es analizar los medios por los que revisamos el código de programación con el propósito de evaluar la efectividad en la detección de errores desde el punto de vista del gestor del proyecto de software dentro de los proyectos que suministran programas críticos bajo el punto de vista de la seguridad.

En el ejemplo planteado, el propósito es la evaluación, la perspectiva es la eliminación de defectos bajo el punto de vista del gestor de proyectos dentro del entorno de aplicaciones en los que la seguridad es un aspecto crítico. Otro ejemplo es:

Nosotros deseamos examinar la efectividad de las normas de programación que usamos para el lenguaje de programación C++, con el fin de determinar si son efectivos en la producción de software, que pueda ser reutilizado dentro de otros proyectos. En particular, estamos interesados en lo que se necesita con el fin de organizar efectivamente un departamento nuevo encargado de el mantenimiento de una biblio-

teca de software reutilizable. En este estudio se da el software de nuestra área de aplicación principal; que es la de control de inventarios.

Aquí, el propósito es la mejora de un estándar de programación; la perspectiva es la de la reutilización bajo el punto de vista del personal encargado de la administración de una biblioteca de software reutilizable, y el entorno son todos aquellos proyectos que impliquen funciones de control de inventarios.

Otro ejemplo adicional es el siguiente:

Deseamos examinar el efecto de utilizar un constructor de interfaces gráficas, sobre la mejora de las interfaces hombre-máquina, que producimos para nuestros sistemas de administración. En particular, deseamos examinar cómo puede esto afectar a la facilidad de manejo de estas interfaces por parte del usuario. Un foco de atención principal es cómo percibirán los usuarios de estos sistemas que dichas interfaces han cambiado.

Aquí, el propósito es analizar una herramienta con el objetivo de determinar una mejora con respecto a la facilidad de uso, bajo el punto de vista del usuario dentro del contexto de los sistemas administrativos. Un ejemplo final es el siguiente:

Nuestro objetivo es examinar el proceso de chequeo de módulos de código de forma que podamos utilizar los resultados de las comprobaciones con el fin de predecir el número de defectos de código en comprobaciones futuras. La perspectiva que deseamos tener surge de una preocupación sobre el exceso de errores que se han cometido y que no han sido detectados hasta el chequeo del sistema. Deseamos ver qué factores son importantes que permitan que un programador tome decisiones sobre si un módulo está disponible para ser entregado a los probadores del sistema, o por el contrario requiere una comprobación adicional. Deseamos concentrarnos en nuestro modelo de ciclo de vida actual con programadores que utilizan el lenguaje de programación FORTRAN.

Aquí, el objetivo es la predicción, el análisis de un proceso — el de chequeo de código — la perspectiva es la de reducción de costes a través de una reducción del número de errores residuales, que se deslizan dentro del propio chequeo del sistema. La perspectiva es desde el punto de vista del programador y el entorno el de los proyectos convencionales que utilizan el lenguaje de programación FORTRAN.

La plantilla propósito se utiliza para definir lo que está siendo estudiado: podría ser un proceso, un producto, un estándar o un procedimiento. La plantilla también define lo que se va a hacer, y la razón de hacerlo. La perspectiva tiene el objetivo de asegurar que los objetivos no son demasiado ambiciosos: al final del día el método OPM requerirá la realización de medidas y estas medidas y los datos estadísticos asociados serán sólo efectivos cuanto más grande sea el número de factores dentro de un nivel razonable. La plantilla del entorno tiene una función similar: reduce el factor de espacio y permite que sean hechas comparaciones estadísticas efectivas entre procesos y productos.

Con el fin de terminar esta sección es provechoso analizar el método OPM en acción sobre un pequeño ejemplo, con el siguiente objetivo de proceso:

Analícese el proceso de confección de prototipos con el propósito de evaluar desde el punto de vista del desarrollador, el número de requisitos que se rechazan por el cliente en una última etapa del proyecto.

Nosotros asumiremos un modelo desechable de confección de prototipos en el que se use una tecnología como la de un lenguaje típico de cuarta generación para desarrollar una versión rápida de un sistema que, cuando el cliente lo ha aceptado, se implemente de forma convencional con la eliminación del propio prototipo. Esto plantea un cierto número de preguntas que entonces pueden ser procesadas con el fin de evaluar el proceso de confección de prototipos desechables:

- ¿Qué medida tomaríamos con los requerimientos que se hayan cambiado durante la parte convencional del proyecto?
- ¿Se deben ponderar todos los requisitos de forma igualitaria o son algunos más complejos que otros?
- ¿Qué tipos de cambios en los requisitos debemos considerar? Después de todo, algunos de ellos serán debidos a imperfecciones en el proceso de confección de prototipos mientras que otros podrían tener que ver con factores extraños que pueden ser debidos a cambios en las propias circunstancias del cliente.

Con el fin de aplicar el método OPM, necesitamos un modelo del proceso que esté llevándose a cabo y un modelo de la perspectiva de calidad: el número de cambios en los requisitos que son exigidos por el cliente no provienen generalmente de cambios en sus propias circunstancias.

Supongamos que el modelo para el proceso de confección de prototipos desechables es:

1. Especificar los requisitos.
2. Valorar cada requisito en importancia según términos del cliente.
3. Valorar cada requisito en términos de la complejidad de su descripción.
4. Planificar una serie de reuniones de revisión en las que se presente a través de un prototipo una cierta selección de requisitos donde el gestor del proyecto tome una decisión sobre en qué se basa la selección de una presentación de, por ejemplo, dos horas de duración.

Supongamos un modelo muy simple de perspectiva de calidad, donde cada requisito R_i esté asociado con

una complejidad ponderada C_i . Entonces el tamaño de los requisitos totales será:

$$\sum_{i=0}^n R_i \cdot C_i$$

Supongamos que una cierta proporción p de estos requisitos han sido puestos en cuestión emprendiéndose un chequeo de aceptación por parte del cliente, y que estos requisitos no han sido debidos a cambios en las propias circunstancias del cliente. Entonces, la métrica

$$e = p \cdot \sum_{i=0}^n R_i \cdot C_i$$

representa una cierta medida de la desviación de los requisitos desde el prototipo a lo largo del chequeo de aceptación.

Este valor puede entonces ser comparado con los valores base de otros proyectos de confección de prototipos que tengan un entorno parecido. Si se ha observado una cierta mejora, la siguiente etapa es intentar descubrir cómo se ha logrado esta mejora, por ejemplo, en este proyecto el cliente puede haber enviado el mismo representante para ayudar en las demostraciones del prototipo y por consiguiente ha conseguido una consistencia mayor que faltaba en otros proyectos, o el personal del desarrollo que estaba implicado en el proyecto puede haber estado formado por analistas en lugar de diseñadores, y así haber constituido una más sólida relación con el cliente. Si se observaba un incremento en la métrica e , entonces un análisis similar debería llevarse a cabo en términos de lo que constituían los factores desfavorables que afectaban al proyecto.

Lo ya expuesto, entonces, ha sido un resumen esquemático del proceso OPM. Un punto importante a considerar es que ya que existe un número casi infinito de métricas que pueden usarse para caracterizar un producto de software y un proceso de software existe una necesidad de determinar la forma de seleccionarlás, o dicho de otra forma, cuál de las OPM es el mejor ejemplo. Una vez que esto ha sido tomado en consideración en una empresa, esa empresa puede entonces implicarse en una actividad continua de mejora de procesos, que la coloque en un nivel 4 ó 5 de la Escala de Modelos de Madurez de Capacidades y así distinguirla de aquellas otras que lleguen sólo a niveles 1 ó 2.

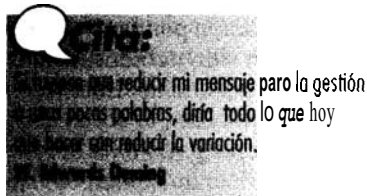
4.8 VARIACIÓN DE LA GESTIÓN: CONTROL DE PROCESOS ESTADÍSTICOS

Debido a que el proceso de software y el producto que tal proceso produce son ambos influenciados por muchos parámetros (por ejemplo: el nivel de habilidad de los realizadores de dichos procesos, la estructura del equipo de software, el conocimiento del cliente, la tecnología que va a ser implementada, las herramientas que serán usadas en la actividad de desarrollo), la métrica elegida para un proyecto o produc-

to no será la misma que otras métricas similares seleccionadas para otro proyecto. En efecto, hay a menudo variaciones significativas en las métricas elegidas como parte del proceso de software.

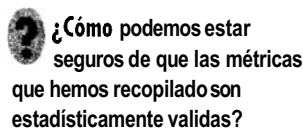
Puesto que la misma métrica de procesos variará de un proyecto a otro proyecto, ¿cómo podemos decir si unos valores de métricas mejoradas (o degradadas) que ocurren como consecuencia de actividades de mejora están

de hecho teniendo un impacto cuantitativo real? ¿Cómo saber si lo que nosotros estamos contemplando es una tendencia estadísticamente válida o si esta «tendencia» es simplemente el resultado de un ruido estadístico? ¿Cuándo son significativos los cambios (ya sean positivos o negativos) de una métrica de software particular?



Se dispone de una técnica gráfica para determinar si los cambios y la variación en los datos de la métrica son significativos. Esta técnica llamada *gráfico de control* y desarrollada por Walter Shewart en 1920¹⁰, permite que los individuos o las personas interesadas en la mejora de procesos de software determine si la dispersión (variabilidad) y «la localización» (media móvil) o métrica de procesos que es *estable* (esto es, si el proceso exhibe cambios controlados o simplemente naturales) o *inestable* (esto es, si el proceso exhibe cambios fuera de control y las métricas no pueden usarse para predecir el rendimiento). Dos tipos diferentes de gráficos de control se usan en la evaluación de los datos métricos [ZUL99]: (1) el gráfico de control de rango móvil (Rm) y (2) el gráfico de control individual.

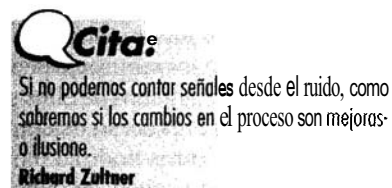
Para ilustrar el enfoque que significa un gráfico de control, consideremos una organización de software que registre en la métrica del proceso los errores descubiertos por hora de revisión, E_r . Durante los pasados 15 meses, la organización ha registrado el E_r para 20 pequeños proyectos en el mismo dominio de desarrollo de software general. Los valores resultantes para E_r están representados en la figura 4.8. Si nos referimos a la figura, E_r varía desde una tasa baja de 1.2 para el proyecto 3 a una tasa más alta de 5.9 para el proyecto 17. En un esfuerzo de mejorar la efectividad de las revisiones, la organización de software proporcionaba entrenamiento y asesoramiento a todos los miembros del equipo del proyecto, comenzando con el proyecto 11.



Richard Zultner proporciona una vista general del procedimiento que se requiere para desarrollar un *gráfico de control de rango móvil* (Rm) para determinar la estabilidad del proceso [ZUL99]:

1. Calcular los rangos móviles: el valor absoluto de las diferencias sucesivas entre cada pareja de puntos de datos... Dibujar estos rangos móviles sobre el gráfico.
2. Calcular la media de los rangos móviles... dibujando ésta («barra Rm») como la línea central del propio gráfico.
3. Multiplicar la media por 3.268. Dibujar esta línea como el *límite de control superior* [LCS]. Esta línea supone tres veces el valor de la desviación estándar por encima de la media.

Usando los datos representados en la figura 4.8 y los distintos pasos sugeridos por Zultner como anteriormente se ha descrito, desarrollamos un gráfico de control Rm que se muestra en la Figura 4.9. El valor (medio) «barra Rm» para los datos de rango móvil es 1.71. El límite de control superior (LCS) es 5.57.



Para determinar si la dispersión de las métricas del proceso es estable puede preguntarse una cuestión muy sencilla: ¿Están los valores de rango móvil dentro del LCS? Para el ejemplo descrito anteriormente, la contestación es «sí». Por consiguiente, la dispersión de la métrica es estable.

El gráfico de control individual se desarrolla de la manera siguiente¹¹:

1. Dibujar los valores de la métrica individual según se describe en la Figura 4.8.
2. Calcular el valor promedio, A_m , para los valores de la métrica.
3. Multiplicar la media de los valores Rm (la barra Rm) por 2.660 y añadir el valor de A_m calculado en el paso 2. Esto da lugar a lo que se denomina *límite de proceso natural superior* (LPNS). Dibujar el LPNS.
4. Multiplicar la media de los valores Rm (la barra Rm) por 2.660 y restar este valor del A_m calculado en el paso 2. Este cálculo da lugar al *límite de proceso natural inferior* (LPNI). Dibujar el LPNI. Si el LPNI es menor que 0.0, no necesita ser dibujado a menos que la métrica que está siendo evaluada tome valores que sean menores que 0.0.
5. Calcular la desviación estándar según la fórmula $(LPNS - A_m)/3$. Dibujar las líneas de la desviación estándar una y dos por encima y por debajo de A_m . Si cualquiera de las líneas de desviación estándar es menor de 0.0, no necesita ser dibujada a menos que la métrica que está siendo evaluada tome valores que sean menores que 0.0.

¹⁰ Debería tenerse en cuenta que aunque el gráfico de control fue desarrollado originalmente para procesos de fabricación es igualmente aplicable a procesos de software.

¹¹ El estudio que sigue es un resumen de los pasos sugeridos por Zultner [ZUL99].

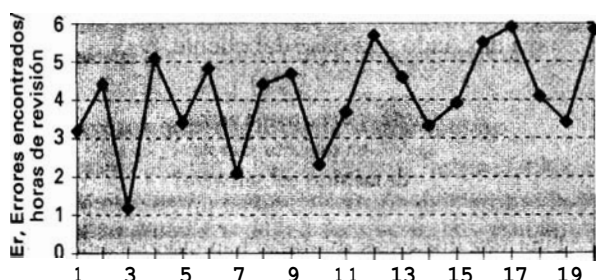


FIGURA 4.8. Datos de métricas para descubrir errores por hora de revisión.

Aplicando estos pasos a los datos representados en la Figura 4.8, se llega a un *gráfico de control individual* según se ve en la figura 4.10.



Referencia Web

Se puede encontrar un Gráfico de Control Común que cubre el tema en alguna dimensión en:
www.sytsma.com/tqmttools/ctlchfprinciples.html

Zultner [ZUL99] revisa cuatro criterios, denominados *reglas de zona*, que pueden usarse para evaluar si los cambios representados por la métrica indican que un proceso está bajo control o fuera de control. Si cualquiera de las siguientes condiciones es verdadera, los datos de la métrica indican un proceso que está fuera de control:

1. Un valor de la métrica individual aparece fuera del LPNS.
2. Dos de cada tres valores de métricas sucesivas aparecen más de dos desviaciones estándar fuera del valor A_r .
3. Cuatro de cada cinco valores de métricas sucesivas aparecen alejados más de una desviación estándar del valor A_r .
4. Ocho valores consecutivos de métrica aparecen todos situados a un lado del valor A_r .

4.9 MÉTRICA PARA ORGANIZACIONES PEQUEÑAS

La amplia mayoría de las organizaciones de desarrollo de software tienen menos de 20 personas dedicadas al software. Es poco razonable, y en la mayoría de los casos no es realista, esperar que organizaciones como éstas desarrollen programas métricos de software extensos. Sin embargo, si que es razonable sugerir que organizaciones de software de todos los tamaños midan y después utilicen las métricas resultantes para ayudar a mejorar sus procesos de software local y la calidad y oportunidad de los productos que realizan. Kautz

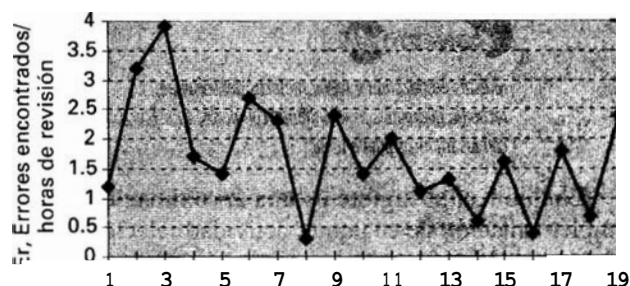


FIGURA 4.9. Gráfico de control de rango móvil (Rm).

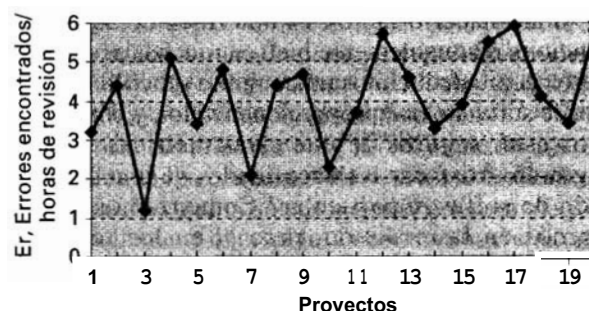


FIGURA 4.10. Gráfico de control individual.

Puesto que todas estas condiciones fallan para los valores mostrados para la figura 4.10, se concluye que los datos de las métricas se derivan de un proceso estable y que se pueden deducir legítimamente a partir de los datos recogidos en la métrica una información que constituye una verdadera tendencia. Si nos referimos a la figura 4.10, puede verse que la variabilidad de E_r decrece a partir del proyecto 10 (esto es, después de un esfuerzo para mejorar la efectividad de las revisiones). Calculando el valor medio para los 10 primeros y los 10 últimos proyectos, puede demostrarse que el valor medio de E_r para los proyectos del 11 al 20, muestran un 29 por 100 de mejora en relación con la E_r de los proyectos 1 al 10. Puesto que el gráfico de control indica que el proceso es estable, parece que los esfuerzos para mejorar la efectividad de la revisión dan sus resultados.

[KAU99] describe un escenario típico que ocurre cuando se piensa en programas métricos para organizaciones pequeñas de software:

Originalmente, los desarrolladores de software acogían nuestras actividades con un alto grado de escepticismo, pero al final las aceptaban debido a que nosotros conseguíamos que nuestras medidas fueran simples de realizar, adaptadas a cada organización y se aseguraba que dichas medidas producían una información válida y útil. Al final, los programas proporcionaban una base para encargarse de los clientes y para la planificación y desarrollo de su trabajo futuro.



Si estás empezando a reunir métricas del software, recuerda guardarlas. Si te entienros con datos, tu esfuerzo con los métricos puede fallar.

Lo que Kautz sugiere es una aproximación para la implementación de cualquier actividad relacionada con el proceso del software: que sea simple; adaptada a satisfacer las necesidades locales y que se constate que realmente añade valor. En los párrafos siguientes examinamos cómo se relacionan estas líneas generales con las métricas utilizadas para negocios pequeños.

«Para hacerlo fácil», es una línea de acción que funciona razonablemente bien en muchas actividades. Pero ¿cómo deducimos un conjunto sencillo de métricas de software que proporcionen valor, y cómo podemos estar seguros de que estas métricas sencillas logran satisfacer las necesidades de una organización de software particular? Comenzamos sin centrarnos en la medición, pero **sí** en los resultados finales. El grupo de software es interrogado para que defina un objetivo simple que requiera mejora. Por ejemplo, «reducir el tiempo para evaluar e implementar las peticiones de cambio». Una organización pequeña puede seleccionar el siguiente conjunto de medidas fácilmente recolectables:

- Tiempo (horas o días) que transcurren desde el momento que es realizada una petición hasta que se complete su evaluación, t_{cola} .
- Esfuerzo (horas-persona) para desarrollar la evaluación, W_{eval} .
- Tiempo (horas o días) transcurridos desde la terminación de la evaluación a la asignación de una orden de cambio al personal, t_{eval} .
- Esfuerzo (horas-persona) requeridas para realizar el cambio, W_{cambio} .
- Tiempo requerido (horas o días) para realizar el cambio, t_{cambio} .
- Errores descubiertos durante el trabajo para realizar el cambio, E_{cambio} .

- Defectos descubiertos después de que el cambio se haya desviado a la base del cliente, D_{cambio} .



¿Cómo puedo derivar un conjunto «simple» de métricas del software?

Una vez que estas medidas han sido recogidas para un cierto número de peticiones de cambio, es posible calcular el tiempo total transcurrido desde la petición de cambio hasta la implementación de dicho cambio y el porcentaje de tiempo consumido por el proceso de colas iniciales, la asignación de cambio y evaluación, y la implementación del cambio propiamente dicho. De forma similar, el porcentaje de esfuerzo requerido para la evaluación y la implementación puede también ser determinado. Estas métricas pueden ser evaluadas en el contexto de los datos de calidad, E_{cambio} y D_{cambio} . Los porcentajes proporcionan un análisis interno para buscar el lugar donde los procesos de petición de cambio se ralentizan y pueden conducir a unos pasos de mejoras de proceso para reducir t_{cola} , W_{eval} , t_{eval} , W_{cambio} , y/o E_{cambio} . Además, la eficiencia de eliminación de defectos (EED) puede ser calculada de la siguiente manera:

$$EED = E_{cambio} / (E_{cambio} + D_{cambio})$$

EED puede compararse con el tiempo transcurrido y el esfuerzo total para determinar el impacto de las actividades de aseguramiento de la calidad sobre el tiempo y el esfuerzo requeridos para realizar un cambio.

Para grupos pequeños, el coste de incorporar medidas y métricas de cálculo oscila entre el 3 y el 8 por 100 del presupuesto del proyecto durante la fase de aprendizaje y después cae a menos del 1 por 100 del presupuesto del proyecto una vez que la ingeniería del software y la gestión de proyectos se hayan familiarizado con el programa de métricas [GRA99]. Estos costes pueden representar una mejora de las inversiones siempre que el análisis derivado a partir de los datos de la métrica conduzcan a una mejora de procesos significativa para la organización del software.

4.10 ESTABLECIMIENTO DE UN PROGRAMA DE MÉTRICAS DE SOFTWARE

El Instituto de Ingeniería del Software (IIS) ha desarrollado una guía extensa [PAR96] para establecer un programa de medición de software dirigido hacia objetivos. La guía sugiere los siguientes pasos para trabajar:

1. Identificar los objetivos del negocio.
2. Identificar lo que se desea saber o aprender.
3. Identificar los subobjetivos.
4. Identificar las entidades y atributos relativos a esos subobjetivos.
5. Formalizar los objetivos de la medición.

6. Identificar preguntas que puedan cuantificarse y los indicadores relacionados que se van a usar para ayudar a conseguir los objetivos de medición.
7. Identificar los elementos de datos que se van a recoger para construir los indicadores que ayuden a responder a las preguntas planteadas.
8. Definir las medidas a usar y hacer que estas definiciones sean operativas.
9. Identificar las acciones que serán tomadas para mejorar las medidas indicadas.
10. Preparar un plan para implementar estas medidas.



Referencia Web

Se puede descargar una guía para la medición del software orientado a objetivos desde: www.sei.tmu.edu

Si se quiere entrar en una discusión más profunda de los pasos anteriores, lo mejor es acudir directamente a la guía ya comentada del IIS (SEI) Instituto de Ingeniería del Software. Sin embargo, merece la pena repasar brevemente los puntos clave de ésta.

Ya que el software, en primer lugar, soporta las funciones del negocio, en segundo lugar, diferencia o clasifica los sistemas o productos basados en computadora, y en tercer lugar puede actuar como un producto en sí mismo, los objetivos definidos para el propio negocio pueden casi siempre ser seguidos de arriba abajo hasta los objetivos más específicos a nivel de ingeniería de software. Por ejemplo, considérese una compañía que fabrica sistemas avanzados para la seguridad del hogar que tienen un contenido de software sustancial. Trabajando como un equipo, los ingenieros de software y los gestores del negocio, pueden desarrollar una lista de objetivos del propio negocio convenientemente priorizados:

1. Mejorar la satisfacción de nuestro cliente con nuestros productos.
2. Hacer que nuestros productos sean más fáciles de usar.
3. Reducir el tiempo que nos lleva sacar un nuevo producto al mercado.
4. Hacer que el soporte que se dé a nuestros productos sea más fácil.
5. Mejorar nuestro beneficio global.



las métricas del software que elijas estarán conducidas por el negocio o por los objetivos técnicos que desees cumplir.

La organización de software examina cada objetivo de negocios y se pregunta: «¿Qué actividades gestionaremos (ejecutaremos) y qué queremos mejorar con estas actividades?» Para responder a estas preguntas el IIS recomienda la creación de una «lista de preguntas-entidad», en la que todas las cosas (entidades) dentro del proceso de software que sean gestionadas o estén influenciadas por la orga-

nización de software sean anotadas convenientemente. Ejemplo de tales entidades incluye: recursos de desarrollo, productos de trabajo, código fuente, casos de prueba, peticiones de cambio, tareas de ingeniería de software y planificaciones. Para cada entidad listada, el personal dedicado al software desarrolla una serie de preguntas que evalúan las características cuantitativas de la entidad (por ejemplo: tamaño, coste, tiempo para desarrollarlo). Las cuestiones derivadas como una consecuencia de la creación de una lista tal de preguntas-entidad, conducen a la derivación de un conjunto de objetivos de segundo nivel (subobjetivos) que se relacionan directamente con las entidades creadas y las actividades desarrolladas como una parte del proceso del software.

Considérese el cuarto objetivo apuntado antes: «Hacer que el soporte para nuestros productos sea más fácil». La siguiente lista de preguntas pueden ser derivadas a partir de este objetivo [PAR96]:

- ¿Contienen las preguntas de cambio del cliente la información que nosotros requerimos para evaluar adecuadamente el cambio y de esa forma realizarle en un tiempo y formas oportunos?
- ¿Cómo es de grande el registro de peticiones de cambio?
- ¿Es aceptable nuestro tiempo de respuesta para localizar errores de acuerdo a las necesidades del cliente?
- ¿Se sigue convenientemente nuestro proceso de control de cambios (Capítulo 9)?
- ¿Se llevan a cabo los cambios de alta prioridad de manera oportuna y sincronizada?

Basándose en estas cuestiones, la organización de software puede deducir los objetivos de segundo nivel (subobjetivos) siguientes: Mejorar el rendimiento del proceso de gestión del cambio. Las entidades de proceso de software y atributos que son relevantes para los propósitos u objetivos más específicos o de segundo nivel son identificados y se definen además las metas u objetivos de medida asociados con dichos atributos.

El IIS [PAR96] proporciona una guía detallada para los pasos 6 al 10 de este enfoque de medida orientado hacia objetivos. En esencia, se aplica un proceso de refinamiento por pasos en el que los objetivos son refinados en preguntas que son a su vez refinadas de forma más detallada en entidades y atributos que por último se analizan en un último paso de forma más minuciosa a nivel de la métrica en sí.

RESUMEN

La medición permite que gestores y desarrolladores mejoren el proceso del software, ayuden en la planificación, seguimiento y control de un proyecto de software, y evalúen la calidad del producto (software) que se produce. Las medidas de los atributos específicos del proceso, del proyecto y del produc-

to se utilizan para calcular las métricas del software. Estas métricas se pueden analizar para proporcionar indicadores que guían acciones de gestión y técnicas.

Las métricas del proceso permiten que una organización tome una visión estratégica proporcionan-

do mayor profundidad de la efectividad de un proceso de software. Las métricas del proyecto son tácticas. Estas permiten que un gestor de proyectos adapte el enfoque a flujos de trabajo del proyecto y a proyectos técnicos en tiempo real.

Las métricas orientadas tanto al tamaño como a la función se utilizan en toda la industria. Las métricas orientadas al tamaño hacen uso de la línea de código como factor de normalización para otras medidas, como persona-mes o defectos. El punto de función proviene de las medidas del dominio de información y de una evaluación subjetiva de la complejidad del problema.

Las métricas de la calidad del software, como métricas de productividad, se centran en el proceso, en el proyecto y en el producto. Desarrollando y analizando una línea base de métricas de calidad, una organización puede actuar con objeto de corregir esas

áreas de proceso del software que son la causa de los defectos del software.

Las métricas tienen significado solo si han sido examinadas para una validez estadística. El gráfico de control es un método sencillo para realizar esto y al mismo tiempo examinar la variación y la localización de los resultados de las métricas.

La medición produce cambios culturales. La recopilación de datos, el cálculo de métricas y la evaluación de métricas son los tres pasos que deben implementarse al comenzar un programa de métricas. En general, un enfoque orientado a los objetivos ayuda a una organización a centrarse en las métricas adecuadas para su negocio. Los ingenieros del software y sus gestores pueden obtener una visión más profunda del trabajo que realizan y del producto que elaboran creando una línea base de métricas —una base de datos que contenga mediciones del proceso y del producto—.

REFERENCIAS

- [ALA97] Alain, A., M. Maya, J. M. Desharnais y S. St. Pierre, «Adapting Function Points to Real-Time Software», *American Programmer*, vol. 10, n.º 11, Noviembre 1997, pp. 32-43.
- [ART85] Arthur, L. J., *Measuring Programmer Productivity and Software Quality*, Wiley-Interscience, 1985.
- [ALB79] Albrecht, A. J., «Measuring Application Development Productivity», *Proc. IBM Application Development Symposium*, Monterey, CA, Octubre 1979, pp. 83-92.
- [ALB83] Albretch, A. J., y J. E. Gaffney, «Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation», *IEEE Trans. Software Engineering*, Noviembre 1983, pp. 639-648.
- [BOE81] Boehm, B., *Software Engineering Economics*, Prentice Hall, 1981.
- [GRA87] Grady, R.B., y D.L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, 1987.
- [GRA92] Grady, R.G., *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, 1992.
- [GRA94] Grady, R., «Succesfully Applying Software Metrics», *Computer*, vol. 27, n.º 9, Septiembre 1994, pp. 18-25.
- [GRA99] Grable, R., et al., «Metrics for Small Projects: Experiences at SED», *IEEE Software*, Marzo 1999, pp. 21-29.
- [GIL88] Gilb, T., *Principles of Software Project Management*, Addison-Wesley, 1998.
- [HET93] Hetzel, W., *Making Software Measurement Work*, QED Publishing Group, 1993.
- [HUM95] Humphrey, W., *A Discipline for Software Engineering*, Addison-Wesley, 1995.
- [IEE93] *IEEE Software Engineering Standards*, Std. 610.12-1990, pp. 47-48.
- [IFP94] *Function Point Counting Practices Manual*, Release 4.0, International Function Point Users Group (IFPUG), 1994.
- [JON86] Jones, C., *Programming Productivity*, McGraw-Hill, 1986.
- [JON91] Jones, C., *Applied Software Costs*, McGraw-Hill, 1991.
- [JON98] Jones, C., *Estimating Software Costs*, McGraw-Hill, 1998.
- [KAU99] Kautz, K., «Making Sense of Measurement for Small Organizations», *IEEE Software*, Marzo 1999, pp. 14-20.
- [MCC78] McCall, J. A., y J. P. Cavano, «A Framework for the Measurement of Software Quality», *ACM Software Quality Assurance Workshop*, Noviembre 1978.
- [PAU94] Paulish, D., y A. Carleton, «Case Studies of Software Process Improvement Measurement», *Computer*, vol. 27, n.º 9, Septiembre 1994, pp. 50-57.
- [PAR96] Park, R. E., W. B. Goethert y W. A. Florac, *Goal Driven Software Measurement-A Guidebook*, CMU/SEI-96-BH-002, Software Engineering Institute, Carnegie Mellon University, Agosto 1996.
- [RAG95] Ragland, B., «Measure, Metric or Indicator: What's the Difference?», *Crosstalk*, vol. 8, n.º 3, Marzo 1995, pp. 29-30.
- [TAL81] Tjima, D., y T. Matsubara, «The Computer Software Industry in Japan», *Computex*, Mayo 1981, p. 96.
- [WHI95] Whitmire, S. A., «An Introduction to 3D Function Points», *Software Development*, Abril 1995, pp. 43-53.
- [ZUL99] Zultner, R. E., «What Do Our Metrics Mean?», *Cutter IT Journal*, vol. 12, n.º 4, Abril 1999, pp. 11-19.

PROBLEMAS Y PUNTOS A CONSIDERAR

- 4.1. Sugiera tres medidas, tres métricas y los indicadores que se podrían utilizar para evaluar un automóvil.
- 4.2. Sugiera tres medidas, tres métricas y los indicadores correspondientes que se podrían utilizar para evaluar el departamento de servicios de un concesionario de automóviles.
- 4.3. Describa, con **sus** propias palabras, la diferencia entre métricas del proceso y del proyecto.
- 4.4. ¿Por qué las métricas del software deberían mantenerse «privadas»? Proporcione ejemplos de tres métricas que deberían ser privadas. Proporcione ejemplos de tres métricas que deberían ser públicas.
- 4.5. Obtenga una copia de [HUM95] y escriba un resumen en una o dos páginas que esquematice el enfoque PSP.
- 4.6. Grady sugiere una etiqueta para las métricas del software. ¿Puede añadir más reglas a las señaladas en la Sección 4.2.1?
- 4.7. Intente completar el diagrama en espina de la Figura 4.3. Esto es, siguiendo el enfoque utilizado para especificaciones «incorrectas», proporcione información análoga para «perdido, ambiguo y cambios».
- 4.8. ¿Qué es una medida indirecta y por qué son comunes tales cambios en un trabajo de métricas de software?
- 4.9. El equipo A encontró 342 errores durante el proceso de ingeniería del software antes de entregarlo. El equipo B encontró 184 errores. ¿Qué medidas adicionales se tendrían que tomar para que los proyectos A y B determinen qué equipos eliminaron los errores más eficientemente? ¿Qué métricas proporcionarían para ayudar a tomar determinaciones? ¿Qué datos históricos podrían ser útiles?
- 4.10. Presente un argumento en contra de las líneas de código como una medida de la productividad del software. ¿Se va a sostener su propuesta cuando se consideren docenas o cientos de proyectos?
- 4.11. Calcule el valor del punto de función de un proyecto con las siguientes características del dominio de información:
 - Número de entradas de usuario: 32
 - Número de salidas de usuario: 60
 - Número de peticiones de usuario: 24

Número de archivos: 8

Número de interfaces externos: 2

Asuma que todos los valores de ajuste de complejidad están en la media.

- 4.12. Calcule el valor del punto de función de un sistema empujado con las características siguientes:

Estructuras de datos interna: 6

Estructuras de datos externa: 3

Número de entradas de usuario: 12

Número de salidas de usuario: 60

Número de peticiones de usuario: 9

Número de interfaces externos: 3

Transformaciones: 36

Transiciones: 24

Asuma que la complejidad de las cuentas anteriores se divide de igual manera entre bajo, medio y alto.

- 4.13. El software utilizado para controlar una fotocopiadora avanzada requiere 32.000 líneas de C y 4.200 líneas de Smalltalk. Estime el número de puntos de función del software de la fotocopiadora.

- 4.14. McCall y Cavano (Sección 4.5.1) definen un «marco de trabajo» de la calidad del software. Con la utilización de la información de este libro y de otros se amplían los tres «puntos de vista» importantes dentro del conjunto de factores y de métricas de calidad.

- 4.15. Desarrolle sus propias métricas (*no* utilice las presentadas en este capítulo) de corrección, facilidad de mantenimiento, integridad y facilidad de uso. Asegúrese de que se pueden traducir en valores cuantitativos.

- 4.16. ¿Es posible que los desperdicios aumenten mientras que disminuyen defectos/KLDC? Explíquelo.

- 4.17. ¿Tiene algún sentido la medida LDC cuando se utiliza el lenguaje de cuarta generación? Explíquelo.

- 4.18. Una organización de software tiene datos EED para 15 proyectos durante los 2 últimos años. Los valores recogidos son: 0.81, 0.71, 0.87, 0.54, 0.63, 0.71, 0.90, 0.82, 0.61, 0.84, 0.73, 0.88, 0.74, 0.86, 0.83. Cree Rm y cuadros de control individuales para determinar si estos datos se pueden utilizar para evaluar tendencias.

OTRAS LECTURAS Y FUENTES DE INFORMACIÓN

La mejora del proceso del software (MPS) ha recibido una significativa atención durante la pasada década. Puesto que la medición y las métricas del software son claves para conseguir una mejora del proceso del software, muchos libros sobre MPS también tratan las métricas. Otras lecturas adicionales que merecen la pena incluyen:

Burr, A., y M. Owen, *Statistical Methods for Software Quality*, International Thomson Publishing, 1996.

Florac, W. A., y A. D. Carleton, *Measuring the Software Process: Statistical Process Control for Software Process Improvement*, Addison-Wesley, 1999.

Garmus, D., y D. Herron, *Measuring the Software Process: A Practical Guide to Functional Measurements*, Prentice-Hall, 1996.

Kan, S. H., *Metrics and Models in Software Quality Engineering*, Addison-Wesley, 1995.

Humphrey [HUM95], Yeh (*Software Process Control*, McGraw-Hill, 1993), Hetzel [HET93] y Grady [GRA92] estudian cómo se pueden utilizar las métricas del software para proporcionar los indicadores necesarios que mejoren el proceso del software. Putnam y Myers (*Executive Briefing: Controlling Software Development*, IEEE Computer Society, 1996) y Pul-

ford y sus colegas (*A Quantitative Approach to Software Management*, Addison-Wesley, 1996) estudian las métricas del proceso y su uso desde el punto de vista de la gestión.

Weinberg (*Quality Software Management, Volume 2: First Order Measurement*, Dorset House, 1993) presenta un modelo útil para observar proyectos de software, asegurándose el significado de la observación y determinando el significado de las decisiones tácticas y estratégicas. Garmus y Herron (*Measuring the Software Process*, Prentice-Hall, 1996) trata las métricas del proceso en el análisis del punto de función. El Software Productivity Consortium (*The Software Measurement Guidebook*, Thomson Computer Press, 1995) proporciona sugerencias útiles para instituir un enfoque efectivo de métricas. Oman y Pfleeger (*Applying Software Metrics*, IEEE Computer Society Press, 1997) han editado una excelente antología de documentos importantes sobre las métri-

cas del software. Park y otros [PAR96] han desarrollado una guía detallada que proporciona paso a paso sugerencias para instituir un programa de las métricas del software para la mejora del proceso del software.

La hoja informativa *IT Metrics* (Editada por Howard Rubin y publicada por los Servicios de Información Cutter) presenta comentarios útiles sobre el estado de las métricas del software en la industria. Las revistas *Cutter IT Journal* y *Software Development* tienen habitualmente artículos y características completas dedicadas a las métricas del software.

En Internet están disponibles una gran variedad de fuentes de información relacionadas con temas del proceso del software y de las métricas del software. Se puede encontrar una lista actualizada con referencias a sitios (páginas) web que son relevantes para el proceso del Software y para las métricas del proyecto en <http://www.pressman5.com>.