Métricas del Mantenimiento de Software

By:
Miguel-Angel Sicilia

Métricas del Mantenimiento de Software

By:
Miguel-Angel Sicilia

Online:
< http://cnx.org/content/col10583/1.9/ >

C O N N E X I O N S

Rice University, Houston, Texas

This selection and arrangement of content as a collection is copyrighted by Miguel-Angel Sicilia, Verónica De la Morena. It is licensed under the Creative Commons Attribution 2.0 license (http://creativecommons.org/licenses/by/2.0/).
Collection structure revised: January 9, 2009
PDF generated: February 5, 2011
For copyright and attribution information for the modules contained in this collection, see p. 30.

Table of Contents

1 Definición de Mantenibilidad 1
2 Aspectos que influyen en la Mantenibilidad 3
3 Propiedades de la Mantenibilidad 5
4 Estándar ISO 9126 del IEEE y la Mantenibilidad 7
5 Métricas de Mantenibilidad del Software 9
6 Métricas de Mantenibilidad Orientadas al Producto 11
7 Métricas de Complejidad del Software 13
8 Métricas Orientadas a Objetos 19
9 Métricas de la Calidad del Diseño Orientado a Objetos del Software 23
10 Técnica del Índice de mantenibilidad 25
11 Métricas relacionadas con el proceso 27
Index 29
Attributions 30

Chapter 1

Definición de Mantenibilidad[1]

El IEEE[2] (19990) define mantenibilidad como: "La facilidad con la que un sistema o componente software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno".

Esta definición está directamente conectada con la definición del IEEE para mantenimiento del software: "es el proceso de modificar un componente o sistema software después de su entrega para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarlo a cambios en el entorno".

En consecuencia, la mantenibilidad es una característica de calidad del software relacionada con la facilidad de mantenimiento, que nosotros consideraremos como una actividad de mantenimiento.

A mayor mantenibilidad, menores costes de mantenimiento (y viceversa).

La mantenibilidad debe establecerse como objetivo tanto en las fases iniciales del ciclo de vida, para reducir las posteriores necesidades de mantenimiento, como durante las fases de mantenimiento, para reducir los efectos laterales y otros inconvenientes ocultos (y seguir así reduciendo las futuras necesidades de mantenimiento).

La calidad del software es una compleja mezcla de factores que variarán a través de diferentes aplicaciones y según los clientes que las pidan. Dichos factores se pueden dividir en 2 grupos:

- factores que se pueden medir directamente
- factores que se pueden medir sólo indirectamente.

McCall[3] propuso una útil clasificación de factores que afectan a la calidad del software. Estos factores son:

- Corrección → Hasta dónde satisface un programa su especificación y logra los objetivos propuestos por el cliente.
- Fiabilidad → Hasta dónde se puede esperar que un programa lleve a cabo su función con la exactitud requerida.
- Eficiencia → La cantidad de recursos informáticos y de códigos necesarios para que un programa realice su función.
- Integridad → Hasta dónde se puede controlar el acceso al software o a los datos por personas no autorizadas.
- Usabilidad → El esfuerzo necesario para aprender a operar con el sistema, preparar los datos de entrada e interpretar las salidas (resultados) de un programa.
- Facilidad de mantenimiento (mantenibilidad) → El esfuerzo necesario para localizar y arreglar un error en un programa.

[1] This content is available online at <http://cnx.org/content/m17457/1.2/>.

[2] Institute of Electrical and Electronics Engineers. (1990) IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY.IEEE Std. 610.12 (1990) Standard Glossary of Software Engineering Terminology. IEEE Computer Society Press, Los Alamitos, CA.

[3] Roger S. Presuman (2002) Ingeniería del Software. Un enfoque práctico

- Flexibilidad → El esfuerzo necesario para modificar un programa que ya está en funcionamiento.
- Facilidad de prueba → El esfuerzo necesario para probar un programa y asegurarse de que realiza correctamente su función.
- Portabilidad → El esfuerzo necesario para transferir el programa de un entorno hardware/software a otro entorno diferente.
- Reusabilidad → Hasta dónde se puede volver a emplear un programa en otras aplicaciones, en relación al empaquetamiento y alcance de las funciones que realiza el programa.
- Interoperatividad → El esfuerzo necesario para acoplar un sistema con otro.

Hewlett-Packard ha desarrollado un conjunto de factores de calidad del software al que se le ha dado el acrónimo de FURPS (Funcionality, Usability, Reliability, Reformance, Supportability). Los atributos contemplados en cada uno de estos cinco factores son:

- Funcionalidad → Se valora evaluando el conjunto de características y capacidades del programa, la generalidad de las funciones entradas y la seguridad del sistema global.
- Usabilidad → Se valora evaluando el conjunto de características y capacidades del programa, la generalidad de las funciones entregadas y la seguridad del sistema global.
- Fiabilidad → Se evalúa midiendo la frecuencia y gravedad de los fallos, la exactitud de las salidas, el tiempo medio de fallos, la capacidad de recuperación de un fallo y la capacidad de predicción del programa.
- Rendimiento → Se mide por la velocidad del procesamiento, el tiempo de respuesta, consumo de recursos, rendimiento efectivo total y eficacia.
- Capacidad de Soporte → Combina la capacidad de ampliar el programa, adaptabilidad y servicios, así como la capacidad para hacer pruebas, compatibilidad, capacidad de configuración del software, la facilidad de instalación de un sistema y la facilidad con que se pueden localizar los programas.

Los factores de calidad FURPS y atributos descritos pueden usarse para establecer métricas de la calidad para todas las actividades del proceso del software

Chapter 2

Aspectos que influyen en la Mantenibilidad[1]

El IEEE[2] (19990) define mantenibilidad como: “La facilidad con la que un sistema o componente software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno”.

Esta definición está directamente conectada con la definición del IEEE para mantenimiento del software: “es el proceso de modificar un componente o sistema software después de su entrega para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarlo a cambios en el entorno”.

En consecuencia, la mantenibilidad es una característica de calidad del software relacionada con la facilidad de mantenimiento, que nosotros consideraremos como una actividad de mantenimiento.

A mayor mantenibilidad, menores costes de mantenimiento (y viceversa).

La mantenibilidad debe establecerse como objetivo tanto en las fases iniciales del ciclo de vida, para reducir las posteriores necesidades de mantenimiento, como durante las fases de mantenimiento, para reducir los efectos laterales y otros inconvenientes ocultos (y seguir así reduciendo las futuras necesidades de mantenimiento).

Existen unos pocos factores que afectan directamente a la mantenibilidad, de forma que si alguno de ellos no se satisface adecuadamente, ésta se resiente. Los tres más significativos son:

- Proceso de desarrollo: la mantenibilidad debe formar parte integral del proceso de desarrollo del software. Las técnicas utilizadas deben ser lo menos intrusivas posible con el software existente. Los problemas que surgen en muchas organizaciones de mantenimiento son de doble naturaleza: mejorar la mantenibilidad y convencer a los responsables de que la mayor ganancia se obtendrá únicamente cuando la mantenibilidad esté incorporada intrínsecamente en los productos software.
- Documentación: En múltiples ocasiones, ni la documentación ni las especificaciones de diseño están disponibles, y por tanto, los costes del mantenimiento se incrementan debido al tiempo requerido para que un mantenedor entienda el diseño del software antes de poder ponerse a modificarlo. Las decisiones sobre la documentación que debe desarrollarse son muy importantes cuando la responsabilidad del mantenimiento de un sistema se va a transferir a una organización nueva.
- Comprensión de Programas: La causa básica de la mayor parte de los altos costes del MS es la presencia de obstáculos a la comprensión humana de los programas y sistemas existentes. Estos obstáculos surgen de tres fuentes principales:
    - La información disponible es incomprensible, incorrecta o insuficiente.

[1] This content is available online at <http://cnx.org/content/m17452/1.2/>.
[2] Institute of Electrical and Electronics Engineers. (1990) IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY.IEEE Std. 610.12 (1990) Standard Glossary of Software Engineering Terminology. IEEE Computer Society Press, Los Alamitos, CA.

· La complejidad del software, de la naturaleza de la aplicación o de ambos.
· La confusión, mala interpretación o olvidos sobre el programa o sistema.

Dependiendo de cómo se haya construido el software se puede aumentar la mantenibilidad. Los generadores de código, por lo general, no producen un código claro ni fácil de comprender, por lo que el mantenimiento del software así generado es peor. Por otro lado, las técnicas de programación estructurada, la aplicación de metodologías de ingeniería del software y el seguimiento de estándares, permiten la obtención de sistemas o componentes software con menos necesidades de mantenimiento, y en el caso de que se produzcan, mucho más fáciles de llevar a cabo.

Más concretamente, se han identificado los factores específicos que influyen en la mantenibilidad:

- Falta de cuidado en las fases de diseño, codificación o prueba.
- Pobre configuración del producto software.
- Adecuada cualificación del equipo de desarrolladores del software.
- Estructura del software fácil de comprender.
- Facilidad de uso del sistema.
- Empleo de lenguajes de programación y sistemas operativos estandarizados.
- Estructura estandarizada de la documentación.
- Documentación disponible de los casos de prueba.
- Incorporación en el sistema de facilidades de depuración.
- Disponibilidad del equipo (computador y periféricos) adecuado para realizar el mantenimiento.
- Disponibilidad de la persona o grupo que desarrolló originalmente el software.
- Planificación del mantenimiento.

Chapter 3

Propiedades de la Mantenibilidad[1]

El IEEE[2] (19990) define mantenibilidad como: "La facilidad con la que un sistema o componente software puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno".

La mantenibilidad se puede considerar como la combinación de dos propiedades diferentes: La reparabilidad y la flexibilidad.

3.1 Reparabilidad

Un sistema software es reparable si permite la corrección de sus defectos con una cantidad de trabajo limitada y razonable.

La reparabilidad se ve afectada por la cantidad y tamaño de los componentes o piezas. Un producto software que consiste en módulos bien diseñados es más fácil de analizar y reparar que uno monolítico, pero el incremento del número de módulos no implica un producto más reparable, ya que también aumenta la complejidad de las interconexiones entre módulos.

Así pues, se debe buscar un punto de equilibrio con la estructura de módulos más adecuada para garantizar la reparabilidad facilitando la localización y eliminación de los errores en unos pocos módulos.

Un software por mucho que se use no se gasta. Ya le podemos dar veces al botón de aceptar, que, en principio, no debería desgastarse y dejar de funcionar sin motivo aparente.

3.2 Flexibilidad

Un sistema software es flexible si permite cambios para que se satisfagan nuevos requerimientos, es decir, si puede evolucionar. Por su naturaleza inmaterial, es software es mucho más fácil de cambiar o incrementar por lo que respecta a sus funciones que otros productos de naturaleza física, por ejemplo, equipos hardware, pero esta flexibilidad se ve disminuida con cada nueva versión de un producto software, ya que cada versión complica la estructura del software y, por tanto, las futuras modificaciones serán más difíciles.

Aunque esto puede considerarse una generalidad, la aplicación de técnicas y metodologías apropiadas pueden minimizar el impacto en la flexibilidad de cada nueva modificación en el software. Es por esto que la flexibilidad es una característica tanto del producto software como de los procesos relacionados con su construcción. En términos de estos últimos, los procesos deben poderse acomodar a nuevas técnicas de gestión y organización, a cambios en la forma de entender la ingeniería, etc.

[1] This content is available online at <http://cnx.org/content/m17471/1.2/>.

[2] Institute of Electrical and Electronics Engineers. (1990) IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY.IEEE Std. 610.12 (1990) Standard Glossary of Software Engineering Terminology. IEEE Computer Society Press, Los Alamitos, CA.

Chapter 4

Estándar ISO 9126 del IEEE y la Mantenibilidad[1]

ISO 9126 es un estándar internacional para la evaluación del Software. Está supervisado por el proyecto SQuaRE, ISO 25000:2005, el cuál sigue los mismos conceptos.

El estándar está dividido en cuatro partes las cuales dirigen, respectivamente, lo siguiente: modelo de calidad, métricas externas, métricas internas y calidad en las métricas de uso.

El modelo de calidad establecido en la primera parte del estándar, ISO 9126-1. Dicho estándar ha sido desarrollado en un intento de identificar los atributos clave de calidad para el software. El estándar identifica 6 atributos clave de calidad:

- Funcionalidad – El grado en que el software satisface las necesidades indicadas por los siguientes subatributos:
  - Idoneidad
  - Corrección
  - Interoperabilidad
  - Conformidad
  - Seguridad
- Fiabilidad – Cantidad de tiempo que el software está disponible para su uso. Está referido por los siguientes subatributos:
  - Madurez
  - Tolerancia a fallos
  - Facilidad de recuperación
- Usabilidad – Grado en que el software hace óptimo el uso de los recursos del sistema. Está indicado por los siguientes subatributos:
  - Facilidad de comprensión
  - Facilidad de aprendizaje
  - Operatividad
- Eficiencia – Grado en que el software hace óptimo el uso de los recursos del sistema. Está indicado por los siguientes subatributos:
  - Tiempo de uso
  - Recursos utilizados
- Mantenibilidad – Facilidad con que una modificación puede ser realizada. Está indicada por los siguientes subatributos:

[1] This content is available online at <http://cnx.org/content/m17461/1.3/>.

  · Facilidad de análisis
  · Facilidad de cambio
  · Estabilidad
  · Facilidad de prueba

- Portabilidad – La facilidad con que el software puede ser llevado de un entorno a otro. Está referido por los siguientes subatributos:
  · Facilidad de instalación
  · Facilidad de ajuste
  · Facilidad de adaptación al cambio

El atributo Conformidad no está listada arriba ya que se aplica a todas las características. Ejemplos son conformidad a la legislación referente a usabilidad y fiabilidad.

Un atributo es una entidad la cual puede ser verificada o medida en el producto software. Los atributos no están definidos en el estándar, ya que varían entre diferentes productos software.

Un producto software está definido en un sentido amplio como: los ejecutables, código fuente, descripciones de arquitectura, y así. Como resultado, la noción de usuario se amplía tanto a operadores como a programadores, los cuales son usuarios de componentes como son bibliotecas software.

El estándar provee un entorno para que las organizaciones definan un modelo de calidad para el producto software. Haciendo esto así, sin embargo, se lleva a cada organización la tarea de especificar precisamente su propio modelo. Esto podría ser hecho, por ejemplo, especificando los objetivos para las métricas de calidad las cuales evalúan el grado de presencia de los atributos de calidad.

Métricas internas son aquellas que no dependen de la ejecución del software (medidas estáticas).

Métricas externas son aquellas aplicables al software en ejecución.

La calidad en las métricas de uso están sólo disponibles cuando el producto final es usado en condiciones reales.

Idealmente, la calidad interna determina la calidad externa y esta a su vez la calidad en el uso.

Este estándar proviene desde el modelo establecido en 1977 por McCall y sus colegas, los cuales propusieron un modelo para especificar la calidad del software. El modelo de calidad McCall está organizado sobre tres tipos de Características de Calidad:

- Factores (especificar): Ellos describen la visión externa del software, como es visto por los usuarios.
- Criterios (construir): Ellos describen la visión interna del software, con es visto por el desarrollador.
- Métricas (controlar): Ellas son definidas y usadas para proveer una escala y método para la medida.

ISO 9126 distingue entre fallos y no conformidad, siendo un fallo el no cumplimiento de los requisitos previos, mientras que la no conformidad afecta a los requisitos especificados. Una distinción similar es hecha entre la validación y la verificación.

4.1 Utilidad de las normas ISO / IEC 9126

Este estándar está pensado para los desarrolladores, adquirentes, personal que asegure la calidad y evaluadores independientes, responsables de especificar y evaluar la calidad del producto software.

Por tanto, puede servir para validar la completitud de una definición de requisitos, identificar requisitos de calidad de software, objetivos de diseño y prueba, criterios de aseguramiento de la calidad, etc.

La calidad de cualquier proceso del ciclo de vida del software (estándar ISO 12.207) influye en la calidad del producto software que, a su vez, contribuye a mejorar la calidad en el uso del producto.

La calidad del software puede evaluarse midiendo los atributos internos (medidas estáticas o productos intermedios) o atributos externos (comportamiento del código cuando se ejecuta).

Chapter 5

Métricas de Mantenibilidad del Software[1]

Se han propuesto cientos de métricas para el software, pero no todas proporcionan un soporte práctico para el desarrollador de software. Algunas demandan mediciones que son demasiado complejas, otras son tan esotéricas que pocos profesionales tienen la esperanza de entenderlas y otras violan las nociones básicas intuitivas de lo que realmente es el software de alta calidad.

Existen una serie de características que deberían acompañar a las métricas efectivas del software. Dichas características son:

- Simples y fáciles de calcular
- Empírica e intuitivamente persuasivas
- Consistentes y objetivas
- Consistentes en el empleo de unidades y tamaño
- Independientes del lenguaje de programación
- Eficaz mecanismo para la realimentación de calidad

Aunque la mayoría de las métricas de software satisfacen las características anteriores, algunas de las métricas comúnmente empleadas dejan de cumplir una o dos.

Las métricas de mantenibilidad no pueden medir el coste de realizar un cambio particular al sistema software, sino que miden aspectos de la complejidad y la calidad de los programas ya que existe una alta correlación entre la complejidad y la mantenibilidad (a mayor complejidad menor mantenibilidad) y entre la calidad y la mantenibilidad (a mayor calidad mayor mantenibilidad – y viceversa –).

Existen maneras de medir la mantenibilidad para todos los elementos software que están o estarán sometidos a mantenimiento: código, documentos de usuario, documentos de análisis o diseño, etc.

Las métricas del software se pueden clasificar en tres categorías (Kan[2], 2002):

1. Métricas de producto. Estas métricas describen las características del producto que de alguna forma determinan la mantenibilidad, por ejemplo el tamaño, complejidad o características del diseño.
2. Métricas del proceso. Las métricas del proceso pueden ser utilizadas para mejorar el desarrollo y mantenibilidad del software. Algunos ejemplos incluyen la eficacia de eliminar defectos durante el desarrollo, el patrón en el que aparecen los defectos durante las pruebas o el tiempo fijo de respuesta del proceso.
3. Métricas de proyecto. Las métricas de proyecto describen las características y ejecución del proyecto. Por ejemplo, el número de desarrolladores, el patrón de staffing en el ciclo de vida, coste, planificación y productividad del software.

Además, algunas métricas pueden pertenecer a múltiples categorías.

[1] This content is available online at <http://cnx.org/content/m17464/1.2/>.

[2] Kan, S. (2002) Software Quality Metrics Overview. En Metrics and Models in Software Quality Engineering, 2nd Edition. Addison Wesley Professional

Chapter 6

Métricas de Mantenibilidad Orientadas al Producto[1]

Estas métricas describen las características del producto que de alguna forma determinan la mantenibilidad, por ejemplo el tamaño, complejidad o características del diseño.

Las 4 métricas orientadas al producto son:

- La densidad de comentarios en el código
- Métricas de Complejidad.
- El índice de madurez del software (IMS)
- Métricas en Orientación a Objetos: Chidamber & Kemerer

6.1 Densidad de comentarios en el código

Aunque no existen muchas métricas conocidas a este respecto, es significativo para el mantenimiento de un sistema o componente software lo bien documentado que se encuentre. Obviamente, cuantos más comentarios haya en el código fuente, mayor mantenibilidad tendrá el software.

Para observar la densidad de comentarios que hay en el código hay que realizar una inspección del código fuente. Si el código fuente está realizado en Java, una medida fácilmente obtenible es la estudia la proporción de javadocs por número de líneas de código significativas, es decir, líneas de código que contengan sentencias que no sean de comienzo o fin (llaves, en el caso de Java) ni comentarios:

$$\mathbf{Densidad\ comentarios} = \frac{\mathbf{LOCS}}{\mathbf{n\ Javadocs}}$$

Cuanto mayor sea la densidad de comentarios, más mantenible será el software examinado.

6.2 Métricas de Complejidad

Son todas las métricas de software que definen de una u otra forma la medición de la complejidad; Tales como volumen, tamaño, anidaciones, costo (estimación), agregación, configuración, y flujo. Estas son los puntos críticos de la concepción, viabilidad, análisis, y diseño de software.

Los 2 tipos de métrica para calcular la complejidad es:

- Complejidad ciclomática de McCabe[2]
- Ciencia del Software de Halstead[3]

[1] This content is available online at <http://cnx.org/content/m17466/1.8/>.
[2] McCabe,T.J., y A.H. Watson, "Solftware Complexity", Crosstalk.
[3] Halstead, M., "Elements of Software Science", Holland.

6.3 Índice de Madurez del Software (IMS)

El estándar del IEEE 982.1-1988 sugiere un índice de madurez del software (IMS) como métrica específica de mantenimiento. Esta métrica proporciona una indicación de la estabilidad de un producto software. A medida que el IMS se aproxima a 1, el producto comienza a estabilizarse, y por lo tanto, menos esfuerzo de mantenimiento requerirá.

Para calcular el índice hacen falta una serie de medidas anteriores:

- Mt = número de módulos en la versión actual.
- Fm = número de módulos en la versión actual que han sido modificados.
- Fa = número de módulos en la versión actual que han sido añadidos.
- Fe = número de módulos de la versión anterior que se han eliminado en la versión actual.

A partir de estas, el IMS se calcula de la siguiente forma:

$$\mathbf{IMS} = \frac{[\mathbf{Mt} \ - \ (\ \mathbf{Fa} \ + \ \mathbf{Fm} \ + \ \mathbf{Fe}\ )]}{\mathbf{Mt}}$$

6.4 Métricas Orientadas a Objetos

Las métricas OO se centran en métricas que se pueden aplicar a las características de encapsulamiento, ocultamiento de información, herencia y técnicas de abstracción de objetos que hagan única a esa clase.

Chidamber & Kemerer[4] proponen una familia de medidas para desarrollos orientados a objetos:

- Métodos ponderados por clase
- Profundidad árbol de herencia
- Número de descendientes
- Acoplamiento entre clases
- Respuesta para una clase
- Carencia de cohesión en los métodos

[4] Chidamber, S.R., D.P. y C.F.Kemerer, "Management Use of Metrics for Object-Oriented Software: An Exploratory Analysis", IEEE Trans. Software Engineering.

Chapter 7

Métricas de Complejidad del Software[1]

Son todas las métricas de software que definen de una u otra forma la medición de la complejidad; Tales como volumen, tamaño, anidaciones, costo (estimación), agregación, configuración, y flujo. Estas son los puntos críticos de la concepción, viabilidad, análisis, y diseño de software.

Los 2 tipos de métrica para calcular la complejidad es:

- Complejidad ciclomática de McCabe[2]
- Ciencia del Software de Halstead[3]

7.1 Complejidad ciclomática de McCabe

La complejidad ciclomática se basa en el recuento del número de caminos lógicos individuales contenidos en un programa. Para calcular la complejidad del software, Thomas McCabe utilizó la teoría y flujo de grafos. Para hallar la complejidad ciclomática, el programa se representa como un grafo, y cada instrucción que contiene, un nodo del grafo. Las posibles vías de ejecución a partir de una instrucción (nodo) se repesentan en el grafo como aristas. Por ejemplo, el código que se muestra a continuación con 2 sentencias selectivas anidadas genera el siguiente grafo:

```
1 if (condicion){

2  if (condicion){

3      A;

       B;

   } else {

4      C;

       D;

5  }
```

[1] This content is available online at <http://cnx.org/content/m18034/1.8/>.
[2] McCabe,T.J., y A.H. Watson, "Solftware Complexity", Crosstalk.
[3] Halstead, M., "Elements of Software Science", Holland.

```
6 }
```

Si se realizase el grafo, se observaría que se encuentran 3 caminos posibles para llegar de la sentencia 1 a la sentencia 6:

Camino 1 (si ambos IF's son verdad): Sentencias 1, 2, 3, 6

Camino 2 (si el primer IF es verdad y el segundo es falso): Sentencias 1, 4,6

Camino 3 (si el primer IF es falso): Sentencias 1, 6

Este programa tiene una complejidad ciclomática de 3.

La complejidad ciclomática se puede calcular de otras maneras. Se puede utilizar la fórmula:

$$\mathbf{v\,(G)\ =\ e\ -\ n\ +\ 2}$$

donde e representa el número de aristas y n el número de nodos.

Otra forma de calcular la complejidad ciclomática consiste en aplicar la siguiente fórmula:

$$\mathbf{v\,(G)\ =\ \text{nmero de regiones cerradas en el grafo}\ +\ 1}$$

7.1.1 Ejemplo de Calcular la Complejidad Ciclomática

Calcular la complejidad ciclomática del método de ordenación de la Burbuja siguiendo el siguiente código:

```
Public static void bubbleSort2 (Sequence S) {

 Position prec, succ;

 int n = S.size();

 for (int i = 0; i < n; i++) {

  prec = S.first();

  for (int j=1; j < n - i; j++) {

   succ = S.after(prec);

   if (valAtPos(prec) > valAtPos(succ))

    S.swapElements(prec, succ);

   rec = succ;

  }

 }

}
```

El código da como resultado el siguiente grafo:

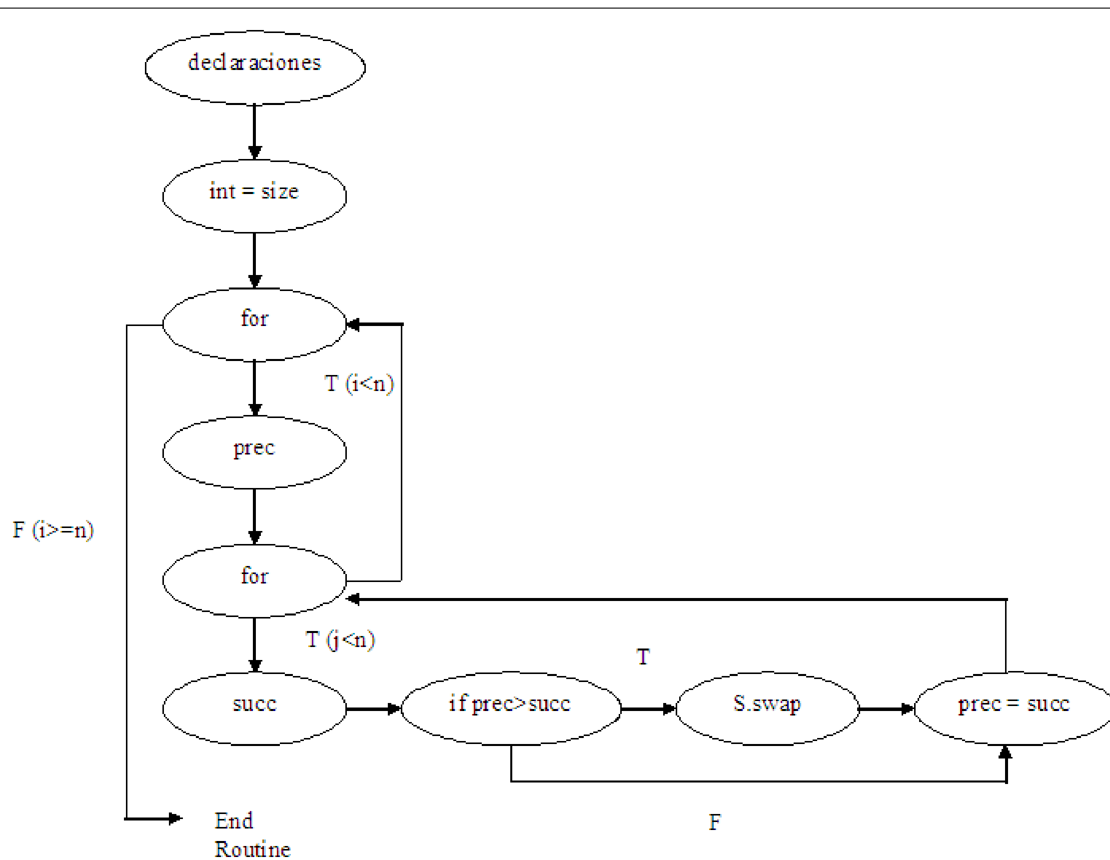

Figure 7.1

$$\mathbf{v(G) = e - n + 2}$$

v(G) = 12 - 10 + 2 = 4
Complejidad ciclomática = 4

7.2 Ciencia del software de Halstead

Durante el final de los años 70 y principios de los 80, Maurice Halstead desarrolla un conjunto de métricas llamadas Halstead Software Science, métricas basadas en el cálculo de palabras clave (reservadas) y variables.

Su teoría está basada en un simple cuenta (muy fácil de automatizar) de operadores y operandos en un programa:

- Los operadores son las palabras reservadas del lenguaje, tales como IF-THEN, READ, FOR,...; los operadores aritméticos +, -, *,..... los de asignación y los operadores lógicos AND, EQUAL TO,....
- Los operandos son las variables, literales y las constantes del programa.

Halstead distingue entre el número de operadores y operandos únicos y el número total de operadores y operando. Por ejemplo, un programa puede tener un READ, siete asignaciones y un WRITE; por lo tanto tiene tres únicos operadores, pero nueve en total operadores, y de manera idéntica se procede con los operandos. Se utiliza la siguiente notación:

- n1 - número de operadores únicos que aparecen en un programa
- N1 - número total de ocurrencias de operadores
- n2 - número de operandos únicos que aparecen en un programa
- N2 - número total de ocurrencias de operandos

Las métricas de la Ciencia del Software para cualquier programa escrito en cualquier lenguaje pueden ser derivadas de estas cuatro cuentas. A partir de ellas han sido elaboradas diferentes medidas para diversas propiedades de los programas, tales como longitud, volumen, etc...

Por ejemplo, consideremos el siguiente trozo de programa:

```
if (N<2){

    A=B*N;

    System.out.println("El resultado es : " + A);

}
```

A partir de aquí se deduce:
N2 = 6 (N, 2, A, B, N, A)
N1 = 6 (if, {}, system.out.println, =, *, <)
n2 = 4 (N, A, B, 2)
n1 = 6 (if, {}, system.out.println, =, *, <)
Halstead permite obtener una medida de la longitud, N, de un programa, que es calculada como:

$$\mathbf{N = N1 + N2}$$

N es una simple medida del tamaño de un programa. Cuanto más grande sea el tamaño de N, mayor será la dificultad para comprender el programa y mayor el esfuerzo para mantenerlo. N es una medida alternativa al simple conteo de líneas de código. Aunque es casi igual de fácil de calcular, N es más sensible a la complejidad que el contar el número de líneas porque N no asume que todas las instrucciones son igual de fácil o de difícil de entender.

La medida de longitud, N, es usada en otra estimación de tamaño de Halstead llamada volumen. Mientras que la longitud es una simple cuenta (o estimación) del total de operadores y operandos, el volumen da un peso extra al número de operadores y operandos únicos. Por ejemplo, si dos programas tienen la misma longitud N pero uno tiene mayor número de operadores y operandos únicos, que naturalmente lo hacen más difícil de entender y mantener, este tendrá un mayor volumen. La fórmula es la siguiente:

$$\mathbf{volumen\ V = N\ x\ log2\,(n)}$$

donde $\mathbf{n = n1 + n2}$

El esfuerzo es otra medida estudiada por Halstead que ofrece una medida del trabajo requerido para desarrollar un programa. Desde el punto de vista del mantenimiento, el esfuerzo se puede interpretar como una medida del trabajo requerido para comprender un software ya desarrollado.

La fórmula es la siguiente:
esfuerzo

$$\mathbf{E = \frac{V}{L}}$$

donde el volumen V es dividido por el nivel del lenguaje L. Éste indica si se está utilizando un lenguaje de alto o bajo nivel. Por ejemplo, una simple llamada a un procedimiento podría tener un valor L de 1; el COBOL podría tener 0,1 y el ensamblador podría tener un L de 0,01. Así pues el esfuerzo aumenta proporcionalmente con el volumen, pero decrece con la utilización de lenguajes de alto nivel.

Atendiendo a varios estudios empíricos, el esfuerzo, E, es incluso una medida mejor de la entendibilidad (comprensión) que N.

7.2.1 Ejemplo para calcular las medidas de Halstead

Calcular las medidas de Halstead de longitud y esfuerzo para el siguiente algoritmo:

```
{
  for (i=2;i<=n;i++)
    for (j=1;j<=i;j++)
      if (x[i] < x[j])
      {
        aux = x[i];
        x[i] = x[j];
        x[j] = aux;
      }
}
```

Para realizar los cálculos de longitud y el volumen, vamos a realizar antes otros cálculos:
Operadores:
{..} → 2
for(;;) → 2
= → 5
if → 1
; → 3
(..) → 1
< → 1
< = → 2
++ → 2
[] → 4
El número total de operadores (n1) son 10 y la cantidad de operadores (N1) que hay son 23.
Operandos:
i → 7
n → 1
j → 6
x → 6
aux → 2

El número total de operandos (n2) son 5 y la cantidad total (N2) son 22.
Por tanto, la longitud es

$$\mathbf{N = N1 + N2}$$

N = 23 + 22 = 45
El volumen es

$$\mathbf{V = N \; x \; log2\,(n)}$$

V = 45 x log2 (10+5)= 175.8

Chapter 8

Métricas Orientadas a Objetos[1]

Las métricas orientadas a objetos se centran en métricas que se pueden aplicar a las características de encapsulamiento, ocultamiento de información, herencia y técnicas de abstracción de objetos que hagan única a esa clase.

Chidamber & Kemerer[2] proponen una familia de medidas para desarrollos orientados a objetos:

- Métodos ponderados por clase (MPC): Tamaño y complejidad
- Profundidad árbol de herencia (PAH): Tamaño
- Número de descendientes (NDD): Tamaño, acoplamiento y cohesión
- Acoplamiento entre clases (ACO): Acoplamiento
- Respuesta para una clase (RPC): Comunicación y complejidad
- Carencia de cohesión en los métodos (CCM): Cohesión interna

Estas métricas, en líneas generales, permiten averiguar cuán bien están definidas las clases y el sistema, lo cual tiene un impacto directo en la mantenibilidad del mismo, tanto por la comprensión de lo desarrollado como por la dificultad de modificarlo con éxito.

8.1 Métodos ponderados por clase

Los métodos ponderados por clase[3] asumen que n métodos de complejidad c1,c2,...cn se definen para la clase C. La métrica de complejidad específica que se eligió debe normalizarse de manera que la complejidad nominal para un método toma un valor de 10. MPC = sumatorio de ci para cada i=1 hasta n.

El número de métodos y su complejidad son indicadores razonables de la cantidad de esfuerzo requerido para implementar y verificar una clase. Cuanto mayor sea el número de métodos, más complejo es el árbol de herencia. A medida que crece el número de métodos para una clase dada, más probable es que vuelvan más y más específicos de la aplicación, así que se limita el potencial de reutilización. El MPC debe mantenerse bajo.

8.2 Profundidad del árbol de herencia

Todos los autores hacen notar la necesidad de medir las estructuras hereditarias en términos de profundidad o de densidad de nodos. Dichas jerarquías pueden medirse como la profundidad de cada clase dentro de su jerarquía, es decir, la longitud máxima desde el nodo que representa la clase hasta la raíz del árbol.

[1] This content is available online at <http://cnx.org/content/m17465/1.5/>.

[2] Chidamber, S.R., D.P. y C.F.Kemerer, "Management Use of Metrics for Object-Oriented Software: An Exploratory Analysis", IEEE Trans. Software Engineering.

[3] Weighted methods per class (WMC)

A medida que crece el valor del PAH, es más probable que las clases de niveles inferiores hereden muchos métodos. Esto da lugar a posibles dificultades cuando se intenta predecir el comportamiento de una clase y por lo tanto, mantenerla. Una jerarquía profunda lleva también a una mayor complejidad de diseño. Por otro lado, los valores grandes de PAH implican que se pueden reutilizar muchos métodos, lo que debe ser considerado como un elemento a favor de la mantenibilidad.

Si tenemos el siguiente árbol:

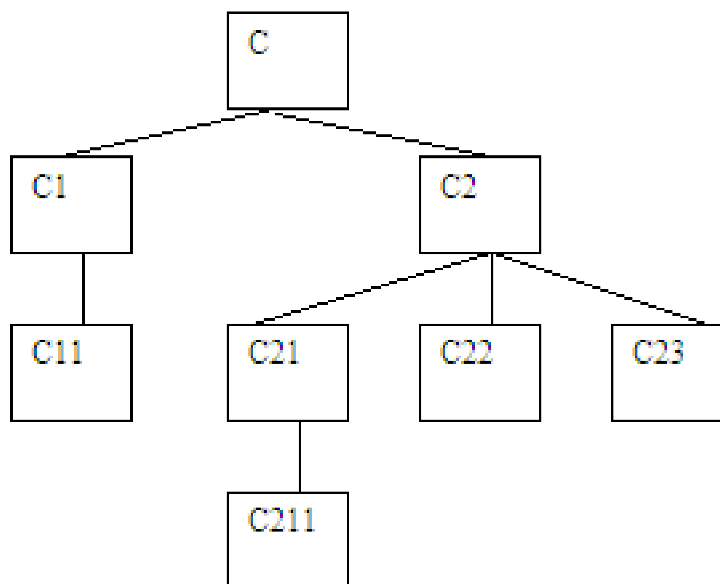

Figure 8.1

El valor del PAH para la jerarquía de las clases mostradas es de 4.

8.3 Número de descendientes

Las subclases que son inmediatamente subordinadas a una clase de la jerarquía de clases se denominan sus descendientes. A medida que crece el número de descendientes[4] NDD, se incrementa la reutilización, pero también implica que la abstracción representada por la clase predecesora se ve diluida. Esto dificulta el mantenimiento, ya que existe la posibilidad de que algunos de los descendientes no sean realmente miembros propios de la clase.

8.4 Acoplamiento entre clases

Para una clase determinada el acoplamiento entre clases[5] se define como el número de otras clases con las cuales está "acoplada". Es por lo tanto una medida del fan-out, esto es, del número de colaboradores (clases que se utilizan desde esa). Sistemas en los cuales una clase tiene un alto ACO y todas las demás tienen

[4]Number of children (NOC)
[5]Coupling between objects (CBO)

valores próximos a cero indican una estructura no orientada a objetos, con una clase principal dirigente. Por el contrario, la existencia de muchas clases con un ACO grande indica que el diseñador ha afinado demasiado la "granularidad" del sistema. Ninguna de las dos situaciones es deseable para la mantenibilidad: la primera situación hace complejo el mantenimiento de la clase con gran ACO y en la segunda situación la complejidad para entender el flujo del programa complica el mantenimiento.

8.5 Respuesta para una clase

La respuesta para una clase (RPC)[6] mide tanto la comunicación interna como la externa. Esta métrica captura el tamaño del conjunto de respuesta para una clase. Este conjunto de respuesta para una clase consiste en todos los métodos llamados por los métodos locales. Definimos RPC como el número de métodos locales más el número de métodos llamados por los métodos locales.

Otra definición que podemos dar del conjunto de respuesta de una clase es la siguiente: "conjunto de métodos que pueden ser ejecutados potencialmente en respuesta a un mensaje recibido por un objeto de esa clase". La RPC se definiría como el número de métodos existentes en el conjunto de respuesta.

Hay que precisar que no se discrimina entre dos mensajes enviados a un mismo método pero desde diferentes partes de la clase.

A medida que el RPC crece, el esfuerzo necesario para las verificaciones y pruebas crece, ya que la complejidad global de diseño de la clase crece también.

8.6 Carencia de cohesión en los métodos

La cohesión de una clase está caracterizada por cuán estrechamente están relacionados los métodos locales a las instancias de variables locales en una clase. La carencia de cohexión en los métodos (CCM)[7] se define como el número de conjuntos disjuntos de métodos locales.

Unos valores elevados para CCM implican que la clase podría diseñarse mejor descomponiéndola en dos o más clases distintas. Aun cuando existen casos en que es justificable un valor elevado de CCM, es deseable mantener un elevado grado de cohesión, esto es, mantener un valor bajo para CCM.

[6] Response for class (RFC)
[7] Lack of cohesion in methods (LCOM)

Chapter 9

Métricas de la Calidad del Diseño Orientado a Objetos del Software[1]

La calidad del software puede ser entendida como el grado con el cual el usuario percibe que el software satisface sus expectativas. El tipo y número de actividades de garantía de calidad que es necesario adoptar en un proyecto o en una organización depende del tamaño y complejidad de los productos software que se estén desarrollando. También influyen otros factores, como pueden ser el tipo de proceso de desarrollo de software o los métodos y herramientas utilizados, la estructura organizativa de la organización, la motivación del personal, entre otros. Según el modelo de calidad ISO 9126, la calidad de un proceso contribuye a mejorar la calidad del producto, y, a su vez, la calidad del producto contribuye a mejorar la calidad en uso. La finalidad de la calidad en uso es medir la efectividad, productividad, seguridad y la satisfacción de los usuarios (pertenecientes a perfiles determinados) que interactúan con el producto en escenarios específicos de uso.

Robert Martin[2], en 1995, estableció un conjunto de métricas para medir la calidad de los diseños orientados a objetos. En términos de la interdependencia entre los subsistemas del diseño (paquetes en Java) partiendo de la base de que, aunque las interrelaciones son necesarias, los diseños con subsistemas fuertemente interrelacionados son muy rígidos, poco reutilizables y difíciles de mantener. Martin propone una medida para cada paquete, que llama inestabilidad (I):

$$\mathbf{I} = \frac{\mathbf{Ce}}{\mathbf{Ca} + \mathbf{Ce}} \tag{9.1}$$

Donde Ce es el número de clases dentro del paquete que dependen de otras clases de otros paquetes; Ca es el número de clases de otros paquetes que tienen una dependencia con clases del paquete.

La medida de inestabilidad está entre 0 y 1. Cuando es 1, significa que ningún paquete depende del paquete que se está midiendo, y por el contrario este paquete si depende de otros paquetes, siendo un paquete inestable: es no responsable y dependiente. La falta de paquetes que dependan de él no le ofrece razones para no cambiar, y los paquetes de los que depende pueden darle amplias razones para cambiar.

Por el contrario, cuando la inestabilidad es nula, I=0, significa que uno o varios paquetes dependen de él, pero él no depende de nadie. Es responsable e independiente, convirtiéndose en un paquete completamente estable. Los módulos que dependen de él hacen fuerza para que sea difícil de cambiar, y como él no depende de otros paquetes no se ve forzado a cambiar.

El principio de las dependencias estables dice que la métrica I de un paquete debe ser mayor que las métricas I de los paquetes de los que él depende, esto es la métrica I debe decrecer en la dirección de la dependencia.

[1] This content is available online at <http://cnx.org/content/m17463/1.4/>.

[2] Martin, R. (1995) OO Design Quality Metrics: An Analysis of Dependencies. Report on Object Analysis&Design, vol. 2 (3).

Se debe tener presente que no es deseable que todos los paquetes sean completamente estables, porque el sistema no podría modificarse.

9.1 Ejemplo de cálculo de la estabilidad

En el ejemplo de la figura siguiente, se va a proceder a calcular la estabilidad del paquete que se encuentra en el centro de la figura. Se tienen 4 clases externas al paquete que tienen dependencia de las clases internas (la clase A se deriva de la clase e1, la clase B contiene objetos de la clase 2, la clase C contiene objetos de la clase 2, y la clase E se deriva de la clase 1), por lo tanto, se tiene que Ca=4.

Por otra parte se tiene que existen 3 relaciones de las clases interiores del paquete que tienen como destino clases exteriores al paquete, es decir, existen 3 dependencias de las clases interiores del paquete con clases exteriores (la clase 1 contiene objetos de la clase G, la clase 1 se deriva de la clase H, y la clase 2 tiene una relación de asociación con la clase I), por lo tanto Ce=3. Así, I=3/7. Por tanto, está casi en el límite de la inestabilidad.

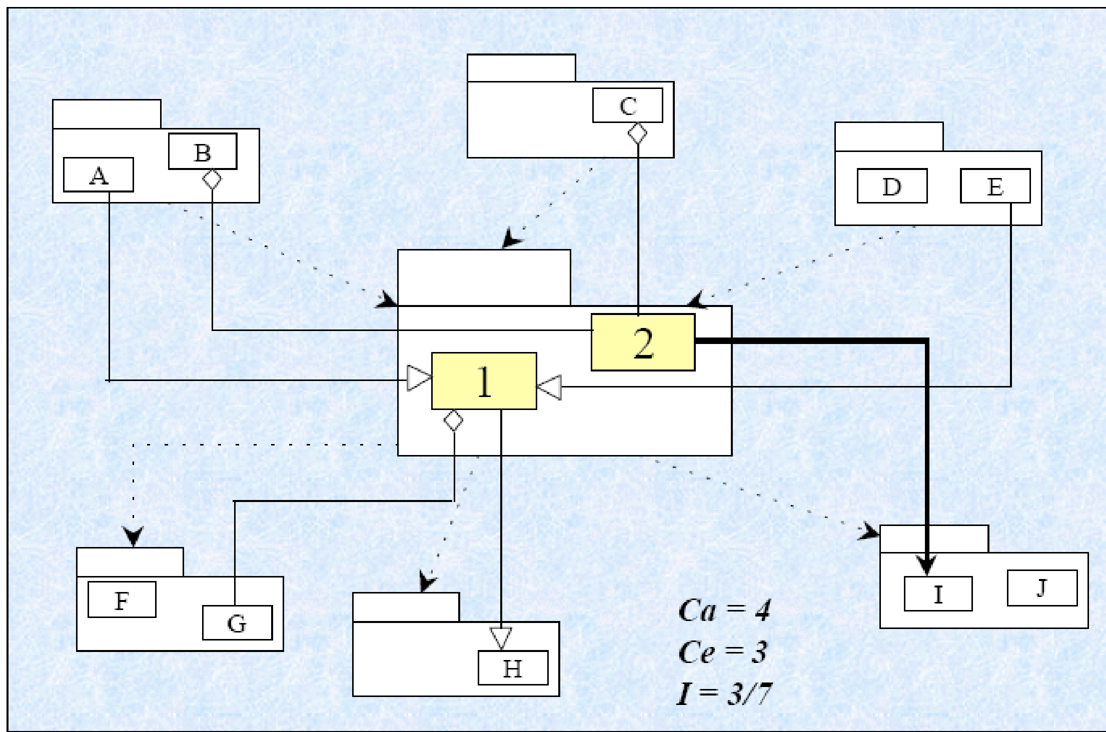

Figure 9.1

Chapter 10

Técnica del Índice de mantenibilidad[1]

El Índice de Mantenibilidad (IM) mide la facilidad de mantenimiento del producto considerado.

Toda acción de mantenimiento puede dividirse en 3 tareas:

- comprensión de los cambios que deben hacerse
- realización de las modificaciones necesarias
- pruebas de los cambios realizados

Welker[2], en 1995, definió el concepto de índice de mantenibilidad IM para intentar cuantificar la mantenibilidad de un sistema. Este concepto ha sido perfeccionado por múltiples autores a lo largo de estos años.

El IM de un conjunto de programas más básico es un polinomio de la siguiente forma:

$$\mathbf{IM = 171 - 5.2 * ln\,(aveV) - 0.23 * aveV\,(g') - 16.2 * ln\,(aveLOC) + 50 * sin\,(sqrt\,(2.4 * perCM))}$$

Donde aveV es la media del volumen V por módulo según Halstead, aveV(g') es la media de la complejidad cilclomática por módulo, aveLOC es la media del número de líneas de código por módulo y perCM es la media porcentual de líneas de código comentadas.

Cuanto mayor sea el IM, mayor mantenibilidad tendrá el sistema.

[1]This content is available online at <http://cnx.org/content/m17473/1.3/>.

[2]Welker, K.D. y Oman, P.W. (1995) Software Maintainability Metrics Models in Practice. Crosstalk, Journal of Defense Software Engineering 8, 11 (November/December 1995), pp. 19-23.

Chapter 11

Métricas relacionadas con el proceso[1]

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. 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 si.

La medición del proceso implica las mediciones de las actividades relacionadas con el software siendo algunos de sus atributos típicos el esfuerzo, el coste y los defectos encontrados. Las métricas permiten tener una visión profunda del proceso de software que ayudará a tomar decisiones más fundamentadas, ayudan a analizar el trabajo desarrollado, conocer si se ha mejorado o no con respecto a proyectos anteriores, ayudan a detectar áreas con problemas para poder remediarlos a tiempo y a realizar mejores estimaciones.

Para mejorar un proceso se deben medir los atributos del mismo, desarrollar métricas de acuerdo a estos atributos y utilizarlas para proporcionar indicadores que conduzcan la mejora del proceso. Los errores detectados antes de la entrega del software, la productividad, recursos y tiempo consumido y ajuste con la planificación son algunos de los resultados que pueden medirse en el proceso, así como las tareas específicas de la ingeniería del software.

Las métricas del proceso se caracterizan por:

- El control y ejecución del proyecto.
- Medición de tiempos del análisis, diseño, implementación, implantación y postimplantación.
- Medición de las pruebas (errores, cubrimiento, resultado en número de defectos y número de éxito).
- Medición de la transformación o evolución del producto.

Estas métricas evalúan el proceso de fabricación del producto correspondiente. Algunos ejemplo clásicos de este tipo de métricas son el tiempo de desarrollo del producto, el esfuerzo que conlleva dicho desarrollo, el número y tipo de recursos empleados (personas, máquinas,...), el coste del proceso, etc.

El tiempo requerido para completar un proceso en particular (tiempo total del proyecto, por ingeniero, por actividad, etc) es un indicador de la mantenibilidad del sistema a tener en cuenta. Aunque no se puede generalizar, cuanto mayor es el tiempo total y por ingeniero para desarrollar un sistema mayor será su complejidad y por lo tanto más difícil será de mantener.

De la misma manera, cuantos mayores sean los costes requeridos para un proceso en particular (esfuerzo en personas-día, costes de viajes, recursos de hardware), menor será la mantenibilidad del sistema.

Además, de estas, el número de defectos descubiertos durante la fase de pruebas y las métricas relacionadas.

[1] This content is available online at <http://cnx.org/content/m17467/1.5/>.

11.1 Defectos descubiertos durante las pruebas

El número de defectos encontrados durante la fase de pruebas una vez que el código está integrado está correlacionado positivamente con el número de defectos del software que se encontrarán durante la explotación del mismo.

Un gran número de defectos durante las pruebas indica que durante la fase de desarrollo se han cometido muchos errores, y, salvo un gran esfuerzo en la fase de pruebas, parte de estos errores se arrastrarán hasta que el sistema esté en explotación.

11.2 Métricas relacionadas con las reparaciones de procesos

Una posible manera de medir la mantenibilidad de un software durante su proceso de construcción es medir la frecuencia de fallos debidos a efectos laterales producidos después de una modificación (X):

$$\mathbf{X} = \frac{\mathbf{1} - \mathbf{A}}{\mathbf{B}}$$

siendo A el número de fallos debidos a efectos laterales detectados y corregidos y B= número total de fallos corregidos.

Cuanto mayor sea X es predecible que más difícil será de mantener en el futuro el sistema.

Index of Keywords and Terms

Keywords are listed by the section with that keyword (page numbers are in parentheses). Keywords do not necessarily appear in the text of the page. They are merely associated with that section. *Ex.* apples, § 1.1 (1) **Terms** are referenced by the page they appear on. *Ex.* apples, 1

C Calidad del Software, § 1(1), § 4(7)
Chidamber & Kemerer, § 8(19)
Complejidad, § 7(13)

H Halstead, § 6(11), § 7(13)

M Mantenibilidad, § 1(1), § 3(5), § 4(7), § 5(9)
Mantenibilidad del Software, § 2(3)
Mantenimiento del Software, § 1(1)
McCabe, § 6(11), § 7(13)
Métricas, § 11(27)
Métricas del Software, § 5(9)

R Robert Martin, § 9(23)

W Welker, § 10(25)

Í Índice de Madurez del Software, § 6(11)
Índice de Mantenibilidad, § 10(25)

Attributions

Collection: *Métricas del Mantenimiento de Software*
Edited by: Miguel-Angel Sicilia
URL: http://cnx.org/content/col10583/1.9/
License: http://creativecommons.org/licenses/by/2.0/

Module: "Definición de Mantenibilidad"
By: Miguel-Angel Sicilia
URL: http://cnx.org/content/m17457/1.2/
Pages: 1-2
Copyright: Miguel-Angel Sicilia, Verónica De la Morena
License: http://creativecommons.org/licenses/by/2.0/

Module: "Aspectos que influyen en la Mantenibilidad"
By: Miguel-Angel Sicilia
URL: http://cnx.org/content/m17452/1.2/
Pages: 3-4
Copyright: Miguel-Angel Sicilia, Verónica De la Morena
License: http://creativecommons.org/licenses/by/2.0/

Module: "Propiedades de la Mantenibilidad"
By: Miguel-Angel Sicilia
URL: http://cnx.org/content/m17471/1.2/
Page: 5
Copyright: Miguel-Angel Sicilia, Verónica De la Morena
License: http://creativecommons.org/licenses/by/2.0/

Module: "Estándar ISO 9126 del IEEE y la Mantenibilidad"
By: Miguel-Angel Sicilia
URL: http://cnx.org/content/m17461/1.3/
Pages: 7-8
Copyright: Miguel-Angel Sicilia, Verónica De la Morena
License: http://creativecommons.org/licenses/by/2.0/

Module: "Métricas de Mantenibilidad del Software"
By: Miguel-Angel Sicilia
URL: http://cnx.org/content/m17464/1.2/
Page: 9
Copyright: Miguel-Angel Sicilia, Verónica De la Morena
License: http://creativecommons.org/licenses/by/2.0/

Module: "Métricas de Mantenibilidad Orientadas al Producto"
By: Miguel-Angel Sicilia
URL: http://cnx.org/content/m17466/1.8/
Pages: 11-12
Copyright: Miguel-Angel Sicilia, Verónica De la Morena
License: http://creativecommons.org/licenses/by/2.0/

Module: "Métricas de Complejidad del Software"
By: Miguel-Angel Sicilia
URL: http://cnx.org/content/m18034/1.8/
Pages: 13-18
Copyright: Miguel-Angel Sicilia, Verónica De la Morena
License: http://creativecommons.org/licenses/by/2.0/

Module: "Métricas Orientadas a Objetos"
By: Miguel-Angel Sicilia
URL: http://cnx.org/content/m17465/1.5/
Pages: 19-21
Copyright: Miguel-Angel Sicilia, Verónica De la Morena
License: http://creativecommons.org/licenses/by/2.0/

Module: "Métricas de la Calidad del Diseño Orientado a Objetos del Software"
By: Miguel-Angel Sicilia
URL: http://cnx.org/content/m17463/1.4/
Pages: 23-24
Copyright: Miguel-Angel Sicilia, Verónica De la Morena
License: http://creativecommons.org/licenses/by/2.0/

Module: "Técnica del Índice de mantenibilidad"
By: Miguel-Angel Sicilia
URL: http://cnx.org/content/m17473/1.3/
Page: 25
Copyright: Miguel-Angel Sicilia, Verónica De la Morena
License: http://creativecommons.org/licenses/by/2.0/

Module: "Métricas relacionadas con el proceso"
By: Miguel-Angel Sicilia
URL: http://cnx.org/content/m17467/1.5/
Pages: 27-28
Copyright: Miguel-Angel Sicilia, Verónica De la Morena
License: http://creativecommons.org/licenses/by/2.0/

Métricas del Mantenimiento de Software
Métricas de Mantenimiento del Software: orientadas a Producto, orientadas a objetos, de calidad del diseño orientado a objetos, relacionadas con el proceso, entre otras.

About Connexions
Since 1999, Connexions has been pioneering a global system where anyone can create course materials and make them fully accessible and easily reusable free of charge. We are a Web-based authoring, teaching and learning environment open to anyone interested in education, including students, teachers, professors and lifelong learners. We connect ideas and facilitate educational communities.

Connexions's modular, interactive courses are in use worldwide by universities, community colleges, K-12 schools, distance learners, and lifelong learners. Connexions materials are in many languages, including English, Spanish, Chinese, Japanese, Italian, Vietnamese, French, Portuguese, and Thai. Connexions is part of an exciting new information distribution system that allows for **Print on Demand Books**. Connexions has partnered with innovative on-demand publisher QOOP to accelerate the delivery of printed course materials and textbooks into classrooms worldwide at lower prices than traditional academic publishers.