INGENIERÍA INVERSA

1. ESTADO DEL ARTE

En este apartado abordaremos una visión general de las métricas de calidad de software, así como algunas herramientas existentes para la evaluación de las mismas sobre un software puntual.

1.1. Métricas de calidad de Software

Hoy en día se utilizan los sistemas de métricas en tiempo de ejecución para medir tiempos, cuellos de botella en las aplicaciones, medir capacidades máximas, etc. Así, las métricas sirven para cuantificar, controlar, predecir y probar el desarrollo de un software y su mantenimiento. [1]

Los tres objetivos fundamentales de la medición son:(Fenton y Pfleeger, 1997)

- Entender qué ocurre durante el desarrollo y el mantenimiento.
- Controlar qué es lo que ocurre en nuestros proyectos.
- Mejorar nuestros procesos y nuestros productos.

La medición dentro de la ingeniería de software, es un tanto joven, y al día de hoy no existe un consenso claro sobre la definición exacta de los conceptos y la terminología. Se da a continuación la definición del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE, por sus siglas en inglés) :

Medición: proceso por el cual se obtiene una medida.
Medida: valor asignado a un atributo de una entidad mediante una medición.

Además cabe aclarar que la métrica más utilizada y relevante es la llamada Complejidad Ciclomática, que es la medición cuantitativa de la complejidad lógica de un programa [2] como el número de caminos independientes dentro de un fragmento de código. En general el rango que se encuentran los resultados son: de 1 a 10, un programa simple sin riesgo; de 11 a 20 es un riesgo más complejo; de 21 a 50, muy complejo de alto riesgo.

Las métricas para java, al estar basado este en el modelo orientado a objetos, hacen hincapié en el encapsulamiento, la herencia, complejidad de y entre clases y el polimorfismo, ya que son estos conceptos los que distinguen a un programa orientado a objetos de cualquier otro. Por lo tanto, 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 una clase concreta.

Por esto es que en Fernandez,P.Jara [3] se describen un conjunto representativo de métricas orientadas a objetos, cuya correlación con número de defectos o esfuerzos de mantenimiento ya ha sido validada. Contamos así con:

- Métricas a Nivel de Sistema.
- Métricas de Acoplamiento.
- Métricas de Herencia.

- Métricas de Clases.
- Métricas de métodos.

1.2. Transpilación de Código

La transpilación de código es una forma específica de compilación donde como entrada se tiene un código fuente en un lenguaje específico y como salida tiene también un código fuente pero en otro lenguaje distinto.

1.3. Herramientas para evaluación de métricas para programas Java

1.3.1. **JCSC – Java Coding Standard Checker:** Es una aplicación que le permite al usuario comparar un código fuente contra estandares de codificacion definibles, es una herramienta un tanto sencilla, cuyo principal objetivo es definir reglas sobre un código, pero a nivel de métricas solo implementa 2 (cantidad de líneas útiles,complejidad ciclomática)

1.3.2. **CheckStyle:** es una herramienta de desarrollo que se rige por un estándar de codificación. Devolviendo cuales reglas no se cumplen en el código y el grado de severidad de estas. Si bien es útil para informar al programador las posibles fallas que están presentes en su código, tiene un número reducido de métricas.

1.3.3. **JavaNCSS:**es una herramienta que nos permite realizar mediciones sobre el código fuente Java, obteniendo el valor de la métrica utilizada.

1.3.4. **JMT (Java Measurement Tool):** Es una herramienta que permite analizar una clase o un proyecto entero de java, y calcula las siguientes métricas en base de esto:

- Métricas por clase:
  - **DIT** Depth of Inheritance Tree: especifica la máxima longitud del camino entre la clase y la clase padre o raíz.
  - **NOC** Number of Children: especifica el número directo de sucesores de la clase.
  - **WMC** Weighted Methods per Class: especifica el número de métodos de la clase en cuestión.
  - **WAC** Weighted Attributes per Class: especifica el número de atributos de la clase en cuestión.
  - **CBO** Coupling between Object Classes: especifica el número de clases que referencian algún método o atributo de la clase en cuestión.
  - **PIM** Cantidad de métodos públicos.
  - **NMI** Number of Methods inherited: especifica el número de métodos heredados de las clases de las que hereda directamente.
  - **NAI** Number of Attributes inherited: especifica el número de atributos heredados de las clases de las que hereda directamente.
  - **NMO** Cantidad de metodo sobrescritos.
  - **RFC** Response for a class: especifica el número de métodos de la propia clase más el número de métodos externos que utiliza la clase en cuestión.

    - **LOC** Líneas de código.
- Métricas por método:
    - **NOP** cantidad de parámetros.
      LOC Líneas de código.
      asd
- Métricas de herencia:
    - **MIF** Method Inheritance Factor: especifica la relación entre el número de métodos heredados y el número total de métodos.
    - **AIF** Attribute Inheritance Factor: especifica la relación entre el número de atributos heredados y el número total de atributos.
- Métricas generales del Sistema
    - **COF** Coupling Factor: ejecuta la siguiente fórmula: (Total number of couplings) / (n^2-n); siendo n el número de clases definidas en el sistema.
    - **ANM** Average Number of Methods per Class.
    - **ANA** Average Number of Attributes per Class.
    - **ANP** Average Number of Parameters per Method.

1.3.5. **Metrics Eclipse Plugin:** Es un plugin de Eclipse que define un conjunto de métricas permitiendo al usuario especificar el mínimo y el máximo resultado permitido para cada una de ellas.

1.3.6. **RSM Resource Standard Metrics**: RSM es una herramienta basada en comandos, es decir se puede ejecutar desde una consola UNIX. Esto permite que RSM para integrarse sin problemas en Visual Studio, Kawa y otros IDE, como por ejemplo Eclipse. RSM utiliza un archivo de licencia y archivo de configuración en el arranque. El archivo de configuración permite al usuario personalizar el grado de análisis de código fuente. RSM típicamente toma conmutadores de tiempo de ejecución que permite al usuario personalizar la salida deseada.

1.3.7. **SDMetrics:** SDMetrics evalúa diagramas UML y para ello define un gran número de métricas que agrupa en: métricas de clases, métricas de interfaces, métricas de paquetes, métricas de casos de uso, métricas de máquina de estados, métricas de actividad, métricas de componentes y métricas generales de diagramas[http://www.sdmetrics.com/LoM.html].

1.3.8. **SONAR:**Sonar es una aplicación web con las siguientes características: está basada en reglas, alertas, rangos, exclusiones y configuración; permite configuración online, dispone de una base de datos y permite combinar métricas en conjunto. Sonar permite extensiones a través de plugins para abarcar nuevos lenguajes de programación, para añadir nuevas reglas o para añadir nuevas métricas. Inicialmente, cubre el lenguaje de programación Java, sin embargo, ya existen plugins de código libre y comerciales que extienden la herramienta para cubrir lenguajes como Flex, PL/SQL o Visual Basic. .

1.4. Herramientas de Transpilación de código

1.4.1. **Visual Paradigm:** Es una herramienta que permita la creación de un diagrama de clases en base a un código fuente.Para luego cada vez que se genere nuevo código los cambios se fusionaran con modelo UML

1.4.2. **uml-reverse-mapper:** Esta herramienta genera automáticamente un diagrama de clases a partir de su código. Utilizando la reflexión, UML Reverse Mapper escanea sus paquetes que contienen su código. Construye las relaciones de clase y puede generar como un archivo Graphviz .dot, un archivo PlantUML .puml o un archivo Mermaid .mmd.

1.4.3. **Rational rose:** es una herramienta de diseño orientada a objetos, que da soporte al modelado visual, es decir, que permite representar gráficamente el sistema, permitiendo hacer énfasis en los detalles más importantes, centrándose en los casos de uso y enfocándose hacia un software de mayor calidad, empleando un lenguaje estándar común que facilita la comunicación.

2. MARCO TEÓRICO

Las herramientas que hemos investigado ninguna tiene una alta capacidad de medición de las distintas métricas o no son lo suficientemente claras para con el usuario. Una posible mejora será una programa claro y conciso para el usuario y con la información de las siguientes métricas, que hemos considerado como las mejores:

Proporción de métodos ocultos
(Method Hiding Factor –MHF-) [Abreu y Melo, 1996]

Definición MHF se define como la proporción de la suma de las invisibilidades de los métodos en todas las clases entre el número total de métodos definidos en el sistema. La invisibilidad de un método es el porcentaje sobre el número total de clases desde las cuales un método no es visible. En otras palabras, MHF es la proporción entre los métodos definidos como protegidos o privados y el número total de métodos.

$$MHF = \frac{\sum_{i=1}^{TC}\sum_{m=1}^{Md(Ci)}\left(1 - V\left(M_{mi}\right)\right)}{\sum_{i=1}^{TC} Md\left(C_i\right)}$$

donde:

$M_d(C_l)$ es el número de métodos declarados en una clase,

$$V\left(M_{mi}\right) = \frac{\sum_{j=1}^{TC} es\_visible\left(M_{mi}, C_j\right)}{TC - 1},$$

$$es\_visible\left(M_{mi}, C_j\right) = \begin{cases} 1 \Leftrightarrow \mathrm{j} \neq \mathrm{i} \wedge \mathrm{C_j \text{ puede llamar a } M_{mi}} \\ 0 \qquad \text{caso contrario} \end{cases}$$

Es decir, para todas las clases, C_1, … ,C_n, un método cuenta 0 si puede ser usado por otra clase y 1 en caso contrario.

En lenguages como C++ o Java donde existe el concepto de método protegido, $V(M_{mi})$ se cuenta como una fracción entre 0 y 1:

$$V\left(M_{mi}\right) = \frac{DC(C_i)}{TC - 1}$$

TC es el número total de clases en el sistema.

Propósito MHF se propone como una medida de encapsulación, cantidad relativa de información oculta.

Proporción de atributos ocultos.
(Attribute Hiding Factor –AHF-) [Abreu y Melo, 1996]

Definición AHF se define como la proporción de la suma de las invisibilidades de los atributos en todas las clases entre el número total de atributos definidos en el sistema. La invisibilidad de un atributo es el porcentaje sobre el número total de clases desde las cuales un atributo no es visible. En otras palabras, AHF es la proporción entre los atributos definidos como protegidos o privados y el número total de atributos.

$$AHF = \frac{\sum_{i=1}^{TC}\sum_{m=1}^{Ad(Ci)}\left(1 - V\left(A_{mi}\right)\right)}{\sum_{i=1}^{TC} Ad\left(C_i\right)}$$

donde:

$$V\left(A_{mi}\right) = \frac{\sum_{j=1}^{TC} es\_visible\left(A_{mi}, C_j\right)}{TC - 1}$$

$$es\_visible\left(A_{mi}, C_j\right) = \begin{cases} 1 \Leftrightarrow \mathrm{j} \neq \mathrm{i} \wedge \mathrm{C_j \text{ puede llamar a } A_{mi}} \\ 0 \qquad \text{caso contrario} \end{cases}$$

TC es el número total de clases en el sistema.

Propósito AHF se propone como una medida de encapsulación.

Proporción de métodos heredados
(Method Inheritance Factor –MIF-) [Abreu y Melo, 1996]

Definición MIF se define como proporción de la suma de todos los métodos heredados en todas las clases entre el número total de métodos (localmente definidos más los heredados) en todas las clases.

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

donde:

$$M_a(C) = M_d(C) + M_i(C)$$

y:

$M_d(C_i)$ es el número de métodos declarados en una clase

$M_a(C_i)$ es el número de métodos que pueden ser invocados en relación a C_i.

$M_i(C_i)$ es el número de métodos heredados (y no redefinidos C_i).

TC es el número total de clases en el sistema.

Propósito Sus autores proponen a MIF como una medida de la herencia y como consecuencia, una medida del nivel de reuso. También se propone como ayuda para evaluar la cantidad de recursos necesarios a la hora de testear.

Definición AIF se define como la proporción del número de atributos heredados entre el número total de atributos.

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

donde:

$$A_a(C) = A_d(C_i) + A_i(C_i)$$

y:

$A_d(C_i)$ es el número de atributos declarados en una clase.

$A_a(C_i)$ es el número de atributos que pueden ser invocados asociados a C_i.

$A_i(C_i)$ es el número total de atributos heredados (y no redefinidos) en C_i.

TC es el número total de clases en el sistema.

Propósito Al igual que MIF, AIF se considera un medio para expresar el nivel de reusabilidad en un sistema.

Proporción de polimorfismo.
Polymorphism Factor (PF) [Abreu y Melo, 1996]

Definición PF se define como la proporción entre el número real de posibles diferentes situaciones polimórficas para una clase C_i entre el máximo número posible de situaciones polimorfícas en C_i. En otras palabras, el número de métodos heredados redefinidos dividido entre el máximo número de situaciones polimórficas distintas.

$$PF = \frac{\sum_{i=1}^{TC} M_0(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]}$$

Donde:

$$M_d(C_i) = M_n(C_i) + M_o(C_i)$$

y,
$M_n(C_i)$ es el número de métodos nuevos.
$M_o(C_i)$ es el número de métodos redefinidos.
$DC(C_i)$ es el número de descendientes de C_i.
TC es el número total de clases en el sistema.

Propósito PF es una medida del polimorfismo y una medida indirecta de la asociación dinámica en un sistema.

Proporción de acoplamiento.
(Coupling Factor –CF-) [Abreu y Melo, 1996]

Definición CF se define como la proporción entre el máximo número posible de acoplamientos en el sistema y el número real de acoplamientos no imputables a herencia. En otras palabras, indica la comunicación entre clases.

$$CF = \frac{\sum_{i=1}^{TC} \left[\sum_{j=1}^{TC} es\_cliente(C_i, C_j) \right]}{TC^2 - TC}$$

donde:

$$es\_cliente = \begin{cases} 1 \Leftrightarrow C_c \Rightarrow C_s \wedge C_c \neq C_s \\ 0 \quad caso\ contrario \end{cases}$$

La relación cliente-proveedor ($C_c \Rightarrow C_s$) representa que la clase cliente (C_c) contiene al menos una referencia no heredada de la clase proveedor (C_s).
TC es el número total de clases en el sistema.

Propósito El acoplamiento se ve como una medida del incremento de la complejidad, reduciendo la encapsulación y el posible reuso. Por tanto, limita la comprensibilidad y mantenibilidad del sistema.

Profundidad en árbol de herencia.
(Depth of Inheritance Tree –DIT-) [Chidamber y Kemerer, 1994]

Definición DIT mide el máximo nivel en la jerarquía de herencia. DIT es la cuenta directa de los niveles en la jerarquía de herencia. En el nivel cero de la jerarquía se encuentra la clase raíz.

Propósito Chidamber y Kemerer proponen DIT como medida de la complejidad de una clase, complejidad del diseño y el potencial reuso. Esto es debido a que cuanto más profunda se encuentra una clase en la jerarquía, mayor será la probabilidad de heredar un mayor número de métodos.

Número de hijos.
(Number of Children –NOC-) [Chidamber y Kemerer, 1994]

Definición NOC es el número de subclases subordinadas a una clase en la jerarquía, es decir, el número de subclases que pertenecen a una clase.

Propósito Según Chidamber y Kemerer, NOC es un indicador del nivel de reuso, la posibilidad de haber creado abstracciones erróneas y es un indicador del nivel de test requerido.

Falta de cohesión en los métodos.
(Lack of Cohesion in Methods –LCOM-). [Chidamber y Kemerer, 1994]

Definición LCOM establece en qué medida los métodos hacen referencia a atributos.
Considérese una clase C1 con n métodos M_1, M_2, …, M_n. Sea $\{I_i\}$ = el conjunto de variables instancias por el método Mi.
Hay *n* conjuntos tales que $\{I_1\}$,... $\{I_n\}$; Sea $P = \{(I_i, I_j) \mid I_i \cap I_j = \varnothing\}$, y $Q = \{(I_i, I_j) \mid I_i \cap I_j \_ \varnothing\}$. Si todos los conjuntos *n* $\{I_1\}$,... $\{I_n\}$ son ∅, entonces P = ∅.

$$LCOM = \begin{cases} |P| - |Q| & si\ |P| > |Q| \\ 0 & \text{caso contrario} \end{cases}$$

Propósito LCOM es una medida de la cohesión de una clase midiendo el número de atributos comunes usados por diferentes métodos, indicando la calidad de la abstracción hecha en la clase.

Complejidad Ciclomática por Método
(Cyclomatic Complexity per Method-CCM-)

Definición: CCM establece la medida de complejidad de un método contando los diferentes caminos de control existentes en el código y estableciendo distintos rangos para las distintas complejidades posibles ,siendo estas de 1 a 10, un programa simple sin riesgo; de 11 a 20 es un riesgo más complejo; de 21 a 50, muy complejo de alto riesgo.

Propósito: CCM es la medida en que un método se extiende en su funcionamiento y podría perderse la cohesión del mismo. Esta métrica será de vital importancia para mantener una alta cohesión en las clases del sistema.

Líneas de Código de un Método
(Lines of Code of a Method -LOCM-)

Definición: LOCM es el número de líneas compiladas por un método.

Propósito: LOCM tiene como propósito que no existan métodos muy largos, lo cual facilita a la mantenibilidad del código en un futuro

Para implementar esto, nos vamos a valer de una herramienta llamada ANTLR, vamos a estar trabajando con la versión 4. ANTLR es un potente generador de analizadores sintácticos para leer, procesar, ejecutar o traducir texto estructurado o archivos binarios.[6]

Además, ANTLR4 genera el patrón visitor, con el cual podremos recorrer los nodos del árbol generado, y poder ejecutar un código que queramos nosotros por cada uno de esos nodos, con esto seremos capaces de almacenar la información del código que necesitemos para luego procesarla y evaluar la calidad del código valiéndonos de las métricas propuestas.

Para poder realizar la transpiración de código a un archivo xmi para luego generar el diagrama de clases correspondiente se necesita de dos componentes:

1) El metamodelo del diagrama que queremos generar
2) Un generador de código que emite texto usando plantillas

Luego de una exhaustiva búsqueda para herramientas en el generador de código, implementaremos String Templates v4. Este es un motor de plantillas Java para generar salida de texto formateada. StringTemplate divide su plantilla en fragmentos de texto y expresiones de atributos que de forma predeterminada están encerrados entre corchetes. StringTemplate ignora todo lo que está fuera de las expresiones de atributo, tratándolo como solo texto. Basándonos en el metamodelo[ANEXO 1], vamos a construir las plantillas para generar el archivo con extensión xmi.

REFERENCIAS BIBLIOGRÁFICAS

[1] Evaluación de métricas de calidad del software sobre un programa Java.Ana María García Sánchez-2010

[2] Medición para la gestión en Ingeniería del Software Dolado, J. et al. 2000

[3] L. Fernández, P. J. Lara. Proceso y herramientas para la productividad en el asesoramiento y medición de calidad en desarrollos Java. Revista de Procesos y Métricas de las Tecnologías de la Información (RPM) ISSN: 1698-2029 VOL. 1, Nº 2, Agosto 2004, 31-41.

[4] MEDICIÓN EN LA ORIENTACIÓN A OBJETOS - Daniel Rodríguez y Rachel Harrison

[5]Cyclomatic Complexity Density and Software Maintenance Productivity-Geoffrey K. Gill and Chris F. Kemerer

[6] ANTLR4 DOCUMENTATION - https://github.com/antlr/antlr4/blob/master/doc/index.md