Ensayo de investigación

Diagnóstico del perfil de los egresados de un programa de nivel superior para construir código fuente de calidad

Recibido: 05-10-2018 Aceptado: 05-09-2019 (Artículo Arbitrado)

Resumen

El análisis estático del producto software es el proceso de evaluar su código fuente sin ejecutar el programa, para identificar problemas que incluyen defectos potenciales, complejidad innecesaria y áreas que requieren un alto mantenimiento. En este trabajo se realiza el análisis estático del código fuente construido por egresados, pertenecientes a diversas generaciones de un programa educativo en Ingeniería en Computación (IC). Los resultados de este análisis permiten establecer el perfil de fortalezas y debilidades que adquieren los egresados durante su formación en este rubro, lo que representa para el programa educativo, el punto de partida en la definición de las estrategias y acciones enfocadas a mejorar la calidad en la producción del código fuente. El análisis se lleva a cabo por medio de un proceso de medición, que incluye la recolección de métricas para evaluar la calidad del diseño del código fuente, escrito bajo el Paradigma Orientado a Objetos (POO).

Abstract

Software product static analysis is the process of evaluating a source code without running a program in order to identify potential problems such as defects, unnecessary complexity and areas that require high maintenance. In this work a static analysis of the source code created by a group of graduate engineers from a Computer Engineering (CE) major was carried out. The results of this analysis allow for the creation of a strengths and weaknesses profile acquired by graduates during their training in this area, which for the educational program, represents the starting point in the definition of strategies and actions focused on improving the quality of source code production. The analysis is carried out through a measurement process, which includes the collection of metrics to evaluate the quality of the source code design, written under the Object Oriented Paradigm (OOP).

Résumé

L'analyse statique du produit logiciel consiste à évaluer son code source sans exécuter le programme afin d'identifier les problèmes pouvant inclure des défauts potentiels, une complexité inutile et des domaines nécessitant une maintenance élevée. Dans ce travail est réalisée l'analyse statique du code source construit par les diplômés, appartenant à diverses générations d'un programme éducatif en génie informatique (IC). Les résultats de cette analyse permettent d'établir le profil des forces et des faiblesses acquises par les diplômés au cours de leur formation dans ce domaine, ce qui représente, pour le programme éducatif, le point de départ de la définition de stratégies et d'actions visant à améliorer la qualité des services. Production de code source. L'analyse est effectuée via un processus de mesure, qui inclut la collecte de mesures permettant d'évaluer la qualité de la conception du code source, écrit sous le paradigme orienté objet (OOP).

Hugo Enrique Martínez Cortés[1*]
Myriam Karenina Reyes Sánchez[2]

Palabras clave: Análisis estático, código fuente, calidad, métricas, Paradigma Orientado a Objetos.
Keywords: Static analysis, source code, quality, metrics, Object Oriented Paradigm.
Mots-clés: Analyse statique, code source, qualité, métriques, Paradigme Orienté Objet.

Introducción

A lo largo de los últimos años el software ha aumentado de manera significativa su tamaño y complejidad, como resultado de los grandes avances en la tecnología de cómputo, el crecimiento y diversificación de sus dominios de aplicación, así como las necesidades de gestión de la información en organizaciones que presentan estructuras y operaciones cada vez más complejas. De acuerdo a Ogheneovo (2014) y Antinyan, Staron y Sandberg (2017), el crecimiento de la complejidad del software tiene un efecto directo en el incremento en los costos de su mantenimiento. A su vez, el mantenimiento de software se define como la totalidad

[1]Instituto de Computación.
Universidad Tecnológica de la Mixteca

[2]Laboratorios de Posgrado.
Universidad Tecnológica de la Mixteca

Correspondencia:
[*]hugoe@mixteco.utm.mx

de actividades requeridas para la modificación del producto software después de su entrega debido a un problema o la necesidad de mejora. (IEEE Std., 1998; Abran, 2004).

De acuerdo a Erlikh (2000), Favre (2008), así como Dehaghani y Hajrahimi (2013); Christa, Madhusudhan, Suma, y Rao (2017) y Alawad, Panta, Zibran, e Islam (2018), el costo del mantenimiento representa entre el 40% y el 90% del costo total del ciclo de vida del software, de esta manera un alto porcentaje del esfuerzo requerido para la operación exitosa de un producto software en las organizaciones se destina a la ejecución de actividades como la corrección de errores, la modificación e incorporación de funcionalidades, o bien, la optimización de las ya existentes. Estas actividades de mantenimiento del software pueden extenderse durante muchos años o incluso décadas después de la etapa de desarrollo (Kemerer y Slaughter, 1999). Los desarrolladores que realizan actividades de mantenimiento enfrentan diversos problemas o desafíos, uno de los más destacables es la escasa capacidad de mantenimiento que presenta el código fuente. Se estima que de las más de un billón de líneas de código fuente que se encuentran en producción en todo el mundo, cerca del 80% presentan deficiencias en su estructura, contienen parches, o no tienen una adecuada documentación (Li et al., 2006)

El principal desafío de las fábricas lo constituye el factor humano, debido a que se requieren ingenieros de software con las competencias necesarias para desarrollar software que cumpla con los factores de calidad establecidos, lo que resulta un reto complejo para la industria. De acuerdo a LeBlanc et al.(2006), la disponibilidad de ingenieros de software calificados, no ha satisfecho la demanda de la industria, y esto agrava los problemas que se presentan en la ejecución de los proyectos, de tal manera que los sistemas de software están siendo diseñados y construidos por personas con una formación educativa insuficiente, o bien, que no cuentan con la experiencia necesaria.

El problema de la educación

Ante este panorama, la situación de la industria se hace más adversa debido al crecimiento constante de la demanda de profesionistas competentes en las diferentes áreas de la Ingeniería de Software (IS), impulsada por un incremento adicional en la demanda de software que se enfoca a múltiples dominios de aplicación. De acuerdo con Hislop (2009) "*La demanda de desarrolladores de software está creciendo a una tasa extraordinaria en la medida en que el software está siendo utilizado en un conjunto de dominios cada vez más amplio*". Aunado a esto, Hislop también afirma que: "*Las instituciones educativas se encuentran bajo una presión creciente, enfocada en producir ingenieros calificados de software*".

Así, las instituciones educativas se enfrentan constantemente al reto de mejorar el proceso de enseñanza-aprendizaje de la IS, con el objetivo de desarrollar las competencias necesarias en sus estudiantes, y lograr su integración y desempeño satisfactorio en la industria del software. No obstante, ésto no es tarea fácil. En una revisión sistemática realizada por Radermacher y Walia (2013), por ejemplo, se establece que los egresados de los programas educativos en Ciencias de la Computación e IS principalmente, presentan una amplia variedad de deficiencias de conocimiento, que incluyen desde habilidades personales hasta destrezas técnicas. En dicho estudio se define al término "*deficiencia de conocimiento*" de la siguiente manera: "*Cualquier falta de destreza, habilidad, o de conocimientos conceptuales, que presenta un estudiante graduado en relación a las expectativas de la industria o academia*". Esta misma revisión de literatura describe las deficiencias de conocimiento más frecuentes que presentan los egresados universitarios que incluyen entre otras: la comunicación oral, trabajo en equipo, resolución de problemas, pruebas y programación.

En el contexto de la problemática que plantea la definición del contenido curricular idóneo para los programas educativos en Ciencias en Computación e IS, es un hecho que la programación constituye un elemento estructural preponderante en los planes de estudio de estas disciplinas. No obstante, Breuker, Derriks, y Brunekreef (2011) argumentan que en el currículo de dichos programas educativos generalmente se otorga escasa importancia al tema de la calidad del código fuente del software, el cual incluye la valoración de sus propiedades estáticas, como son: su legibilidad, entendimiento, estructura, formato, complejidad, o bien, su redundancia.

En los cursos de programación típicamente se evalúan únicamente los aspectos funcionales o dinámicos del producto software creado por los estudiantes, pero regularmente no se consideran las propiedades estáticas de su código fuente. Xinogalos e Ivanović (2013) estudiaron los resultados de la incorporación del tópico de calidad del código fuente

del software en cursos de programación básica, como parte de programas educativos de nivel licenciatura, y encontraron evidencia preliminar de que este enfoque puede ayudar a los estudiantes a desarrollar de manera sistemática programas de buena calidad, al mismo tiempo que se preparan para su trabajo futuro en la industria de software. En el presente trabajo se realiza una evaluación estática, a través de un proceso de medición que tiene como objetivo establecer un diagnóstico de las fortalezas y debilidades para construir código fuente de calidad bajo el POO, que adquieren los egresados durante su formación en un programa en Ingeniería en Computación. El resultado del análisis, constituye desde la perspectiva del programa educativo, un punto de partida para diseñar estrategias de mejora, tanto en el contenido curricular del programa de estudios, así como en el proceso de enseñanza-aprendizaje.

Trabajos relacionados

Existe evidencia empírica que muestra que los estudiantes de programas educativos de nivel superior presentan deficiencias en la elaboración de código fuente de calidad. Breuker et al. (2011), por ejemplo, realizaron un estudio con estudiantes que finalizaron el primero y segundo año de un programa en Ciencias de la Computación a nivel Licenciatura, y determinaron que ambos grupos de estudiantes crearon código de baja calidad, sin diferencias destacables. En el trabajo de Pribela, Rakić y Budimac (2012) se aplicaron métricas de software en el proceso de evaluación del código fuente de estudiantes, utilizando un sistema de pruebas automatizado, el cual proporciona sugerencias y consejos a los estudiantes en sus soluciones, y ésto mejora su experiencia de aprendizaje para diversos cursos de programación. Además, implementaron una herramienta que calcula los resultados de las métricas de forma automatizada y ofrece a los instructores la flexibilidad de elegir las métricas a aplicar. Por su parte Hashiura, Matsuura y Komiya (2010) reportan el uso de una herramienta de diagnóstico de calidad del código fuente de programas elaborados por estudiantes, como parte de su etapa de formación educativa en el área de programación, a partir de los cursos de nivel básico. La herramienta ofrece retroalimentación a los estudiantes sobre la evaluación realizada, a partir de la cual pueden ejecutar un autoanálisis de sus programas y crear una conciencia de la calidad de su código fuente, con el objetivo final de mejorar sus competencias en este rubro.

A diferencia de los trabajos mencionados previamente, el presente estudio se realizó con el código fuente elaborado por egresados pertenecientes a diversas generaciones del programa educativo, aunado a lo anterior, la muestra de código evaluada corresponde a un producto software implementado en el marco de un proyecto profesional, con un cliente real. Finalmente, en el presente trabajo se aplican métricas ampliamente aceptadas, que miden a nivel de clase y módulo, las propiedades estructurales del código bajo el POO, como son: el tamaño/complejidad, el acoplamiento, la cohesión, la herencia y el polimorfismo.

Análisis estático del código fuente

Las técnicas de análisis dinámico y estático se utilizan para entender y evaluar la arquitectura de un sistema software. En específico, el análisis estático se realiza directamente sobre el código fuente, con el propósito de extraer el modelo arquitectónico, así como las características del código fuente, recuperar la estructura del sistema, analizar la dependencias de los componentes, identificar las llamadas a funciones y detectar problemas de codificación o de seguridad, problemas que incluyen defectos potenciales, complejidad innecesaria y áreas que requieren un alto mantenimiento (Callo Aria et al., 2011; Truong, Roe y Bancroft (2004)). Con el propósito de evaluar la calidad de un producto software de manera objetiva, se lleva a cabo un proceso de medición, que incluye la recolección de métricas con el propósito de caracterizar de manera cuantitativa, los distintos atributos del producto software. De acuerdo a IEEE Std. 610.2 (1990), se define métrica como “una medida cuantitativa del grado en el que un sistema, componente o proceso posee un atributo determinado”. Para ejecutar el proceso de medición en el presente trabajo, se aplicaron los pasos descritos en Sommerville (2005), que se enlistan a continuación:

1. Seleccionar las medidas a realizar. Se deben formular las preguntas que la medición intenta responder y definir las mediciones requeridas para resolver estas preguntas. No se recogen las mediciones que no están relacionadas de forma directa con estas preguntas.
2. Seleccionar los componentes a evaluar. No es necesario o deseable estimar los valores de las métricas de todos los componentes de un sistema software. En algunos casos, para la medición se elige un conjunto representativo de componentes. En otros, se evalúan los componentes particularmente críticos.

3. Medir las características de los componentes. Se miden los componentes seleccionados y se calculan los valores de las métricas. Normalmente. esto comprende procesar la representación del componente (diseño, código. etc.) utilizando una herramienta de recolección de datos.

4. Identificar las mediciones anómalas. Una vez que se obtienen las mediciones de los componentes, se comparan entre sí y con las mediciones previas registradas en una base de datos de mediciones, o bien, se comparan con las medidas recomendadas en la literatura.

5. Analizar los componentes anómalos. Una vez identificados los componentes con valores anómalos para las métricas particulares, se examinan estos componentes para decidir si los valores de la métrica indican que la calidad del componente está en peligro o no es suficiente.

Desarrollo

Seleccionar las medidas a realizar

Ante la problemática que presenta la adecuada formación de ingenieros de software en instituciones de educación superior, la pregunta a responder en este trabajo y en un entorno local, es la siguiente: "*¿Cuál es el perfil de fortalezas y debilidades para elaborar código fuente de calidad, enfocada en la correcta aplicación de los principios y mecanismos del POO, que adquieren los egresados durante su formación en un programa en Ingeniería en Computación (IC)*?"

Se han propuesto diversas métricas para evaluar la calidad de diseño del código fuente bajo el POO, entre las más ampliamente aceptadas se encuentran las métricas CK (Chidamber y Kemerer, 1991; Chidamber y Kemerer, 1994) y las métricas para el diseño orientado a objetos (MOOD, por sus siglas en inglés) (E Abreu y Melo, 1996).

El conjunto de métricas CK de Chidamber y Kemerer (1991,1994) evalúan propiedades estructurales del código como el tamaño/complejidad, el acoplamiento, la cohesión y la herencia. Éstas son:

• WCM (Weighted Method Count): Es la suma de la complejidad ciclomática de cada método en una clase.

• DIT (Depth of Inheritance Tree): Longitud máxima del camino desde la clase hacia la raíz, en la jerarquía de herencia.

• NOC (Number of Children): Número de subclases inmediatas para una clase.

• CBO (Coupling Between Objects): Número de clases con las que una clase base está acoplada. Dos clases están acopladas si una invoca un método o usa un atributo de la otra. No se considera acoplamiento por herencia.

• RFC (Response For a Class): Número de métodos que pueden ser ejecutados en respuesta a un mensaje recibido por un objeto de una clase.

• LCOM (Lack of Cohesion in Methods): Definido como el número de conjuntos disjuntos de métodos locales.

El conjunto de métricas MOOD está integrado por seis métricas que evalúan propiedades estructurales como la herencia, el acoplamiento y el polimorfismo. Éstas son:

• MHD (Method Hiding Factor): Definida como la proporción entre la suma de todas las invisibilidades de todos los métodos definidos en todas las clases, por el número total de métodos de clase en el componente.

• AHF (Attribute Hiding Factor): Definida como la proporción entre la suma de todas las invisibilidades de todos los atributos definidos en todas las clases, por el número total de atributos de clase en el componente.

• MIF (Method Inheritance Factor): Es la proporción entre la suma de métodos heredados (no redefinidos) en todas las clases, por el número total de métodos disponibles para todas las clases en el componente.

• AIF (Attribute Inheritance Factor): es la proporción entre la suma de atributos heredados en todas las clases, por el número total de atributos disponibles para todas las clases en el componente.

• CF (Coupling Factor): es la proporción del número total de acoplamientos, presente en las clases, por el número máximo posible de acoplamientos entre todas las clases en el componente.

• PF (Polymorphism Factor): es la proporción del número de métodos de clase que redefinen métodos heredados, por el número máximo posible de situaciones polimórficas en el componente.

Este conjunto de métricas se ha seleccionado debido a que permiten dar respuesta a la interrogante planteada. En la tabla 1, se describe para cada una de las métricas CK y MOOD, la propiedad del POO sobre la que está diseñada, su medida recomendada, así como el atributo de calidad para el cual es aplicable.

Seleccionar los componentes a evaluar

La muestra para el proceso de medición se obtuvo del código fuente construido por 28 egresados del programa educativo, los cuales finalizaron sus estudios en el periodo comprendido entre los años 2007-2016, es importante resaltar que el 75% de graduados, pertenecen a las generaciones dentro del periodo 2012-2016. Se destaca de igual manera, que los egresados elaboraron el código fuente en el mismo proyecto, es decir, bajo condiciones similares de planeación, organización, dirección y control. Durante la ejecución del proyecto se integraron equipos homogéneos de cinco programadores en promedio, cada equipo compuesto por dos programadores que egresaron antes de 2014 y tres programadores egresados a partir del año 2014. En la tabla 2 se muestra la información técnica del proyecto, relevante para el presente trabajo.

Tabla 1. Resumen de Métricas CK y MOOD (Romero, 2017)

Métrica	Propiedad del diseño orientado a objetos	Medida recomendada	Atributo de calidad
WMC	Complejidad	20 al 25	Fiabilidad, Escalabilidad, Desempeño, Integración
DIT	Herencia	1 al 5	Funcionalidad, Eficiencia, Mantenibilidad, Desempeño, Escalabilidad, Integración
NOC	Herencia	0 al 5	Funcionalidad, Eficiencia, Mantenibilidad, Desempeño, Escalabilidad, Integración
CBO	Acoplamiento	1 a 4	Fiabilidad, Escalabilidad, Mantenibilidad, Integración
RFC	Colaboración, Comunicación	0 al 15	Fiabilidad, Escalabilidad, Mantenibilidad, Integración, Eficiencia
LCOM	Cohesión	deseable 0	Fiabilidad, Escalabilidad, Mantenibilidad, Integración
PF	Polimorfismo	0 al 10%	Fiabilidad, Escalabilidad, Mantenibilidad, Integración, Eficiencia
CF	Acoplamiento	<10%	Fiabilidad, Escalabilidad, Mantenibilidad, Integración
MHF	Encapsulación	12 a 22%	Eficiencia, Mantenibilidad, Escalabilidad
AHF	Encapsulación	> 75%	Eficiencia, Mantenibilidad, Escalabilidad
MIF	Herencia	60 al 80%	Funcionalidad, Eficiencia, Mantenibilidad, Desempeño, Escalabilidad, Integración
AIF	Herencia	50 al 70%	Funcionalidad, Eficiencia, Mantenibilidad, Desempeño, Escalabilidad, Integración

En este proyecto el 100% de los egresados tomaron el curso de programación orientada a objetos en lenguaje Java impartido en el programa educativo, pero al iniciar el proyecto no contaban con conocimientos en el uso de Marcos de Trabajo Java especializados (indicados en la tabla 2), además representó su primera experiencia profesional en el desarrollo de un producto software con un tamaño mayor a 250,000 líneas de código. El 86% tuvo su primera experiencia profesional en el desarrollo de software bajo el lenguaje Java, y para el 60% significó el primer proyecto profesional de desarrollo de software. En este contexto, se establece que el código fuente creado durante el proyecto mencionado, constituye un objeto de estudio válido para realizar el proceso de medición y dar respuesta al cuestionamiento de referencia.

Con el propósito de ejecutar la medición, se realizó el proceso de selección de una muestra de 22 módulos del total disponible en el proyecto, ésto representa un promedio de cuatro módulos programados por cada equipo. Es importante destacar, que la muestra se adquirió durante la fase de desarrollo del producto software. En la tabla 3 se presentan los datos para caracterizar la muestra. En relación al tamaño, contiene un total de 178,883 líneas de código, 1,905 clases y 14,238 métodos. Aunado a ésto, presenta un 13.86% de duplicidad de código, es decir, del total de 178,883 líneas de código fuente, 24,793 corresponden a código repetido. Esta cantidad de líneas duplicadas representan un mayor esfuerzo de mantenimiento, ya que una modificación del código fuente debe aplicarse en todas las réplicas. Finalmente, la muestra contiene un total de 72,219 líneas de comentarios, lo que repre-

Tabla 2. Información técnica del proyecto

Elemento del Proyecto	Descripción
Duración del proyecto	2014-2017
Total de desarrolladores. Egresados y año de egreso.	28 egresados participantes (año de egreso): 1 (2007), 2 (2008), 1 (2009), 1 (2010), 2 (2011), 4 (2012), 10 (2014), 2 (2015) y 5 (2016)
Lenguaje de Programación	Java versión 7
Entorno de Desarrollo Integrado	Eclipse Luna
Marcos de trabajo utilizados	Java Server Faces (PrimeFaces), Spring, MyBatis y JUnit.
Herramienta para la administración y construcción del proyecto.	Maven
Herramienta de Integración Continua	Jenkins
Herramienta de control de versiones para el código fuente del proyecto	Subversion (SVN)
Herramienta para la administración de la calidad del código fuente.	SonarQube

Tabla 3. Datos para caracterizar la muestra del código fuente

Métrica	Dato
Líneas de Código	178,883
Clases	1905
Métodos	14238
Líneas duplicadas	13.86%
Densidad de comentarios	28.70%
Líneas de comentarios	72,219

Tabla 4. Medidas recolectadas para las métricas CK

Métrica	Propiedad del diseño orientado a objetos	Medida recomendada	Medida promedio por clase	Porcentaje de clases que cumplen con la medida recomendada
WMC	Complejidad	20 al 25	11.7	5%
DIT	Herencia	1 al 5	0.64	61%
NOC	Herencia	0 al 5	0.31	99%
CBO	Acoplamiento	1 a 4	1.15	38%
RFC	Colaboración, Comunicación	0 al 15	24.27	63%
LCOM	Cohesión	0	0.42	76%

senta aproximadamente una tercera parte del total de líneas presentes en cada clase. Ésto constituye un apoyo para facilitar el entendimiento del código.

Medir las características de los componentes

SonarQube es una plataforma de software libre utilizada para llevar a cabo el proceso de recolección de métricas en el código fuente. Esta plataforma apoya a las organizaciones en la administración y control de la calidad del código de sus productos software. En el trabajo de Romero (2017), se implementó una extensión para la plataforma SonarQube que recolecta las métricas CK y MOOD para código escrito en lenguaje Java, la cual se utilizó en el presente trabajo.

SonarQube internamente hace uso de 3 herramientas de análisis de código conocidas como CheckStyle, PMD y FindBugs. CheckStyle es una herramienta que valida el apego a los estándares de construcción, así como identifica problemas potenciales en el código fuente. Por su parte, PMD analiza el código fuente para detectar defectos de programación comunes, reconoce los patrones en código que pueden generar defectos. FindBugs es una herramienta que analiza el código generado después de compilar el programa fuente con el objetivo de encontrar defectos potenciales.

Para tener acceso al código fuente en el repositorio del proyecto, se configuró la herramienta SonarQube por medio de la herramienta de control de versiones SVN, y a continuación se recolectaron las métricas.

Los resultados de las métricas CK se presentan en la tabla 4, estas métricas se aplican a nivel de clase. En la tabla se incluye la medida promedio recolectada por clase, así como el porcentaje de clases cuya medición se ajusta al rango recomendado.

Los resultados de las métricas MOOD se presentan en la tabla 5, estas métricas son globales y se calculan a nivel de componentes o módulos.

Tabla 5. Medidas recolectadas para las métricas MOOD

Métrica	Propiedad del diseño orientado a objetos	Medida recomendada	Medida recolectada
PF	Polimorfismo	0 al 10%	28%
CF	Acoplamiento	0 al 10%	46%
MHF	Encapsulación	12 al 22%	29%
AHF	Encapsulación	75% al 100%	56%
MIF	Herencia	60 al 80%	35%
AIF	Herencia	50 al 70%	15%

Identificar las mediciones anómalas

Con el propósito de detectar las mediciones anómalas, se realizó una comparación entre los resultados obtenidos y los valores recomendados en la literatura (véase tabla 1).

A partir de los datos presentes en la tabla 4 para las métricas CK, se puede afirmar que los resultados de las métricas DIT, NOC, RFC y LCOM son adecuados para al menos el 61% de las clases Java evaluadas. Sin embargo, el resultado para las métricas CBO y WMC, se ajusta al valor de referencia únicamente para el 38% y el 5% de las clases, respectivamente. En consecuencia, estas dos métricas requieren una mayor atención, debido a que muestran un alto porcentaje de clases anómalas, con el caso más acentuado en la métrica WMC.

En el caso de las métricas MOOD, tomando como base los datos presentes en la tabla 4 se observa que, para las seis métricas, el resultado no se ajusta a la medida recomendada. Pero se debe matizar en este punto, que la medición recolectada para la métrica MHF, PF y AHF se desvía del valor de referencia en solo 7, 18 y 19 puntos porcentuales, respectivamente. En cambio, existe una mayor desviación en los resultados obtenidos para las métricas CF, AIF y MIF, con un total de 36, 35 y 25 puntos porcentuales.

Con el propósito de establecer si se presenta el mismo patrón de resultados para las métricas CK y MOOD en todos los equipos, se muestra información adicional de las mediciones, que incluye una clasificación por módulo y por el equipo encargado de su construcción.

Tabla 6. Porcentaje de cumplimiento del valor recomendado para cada métrica CK y MOOD, por módulo y equipo

Métricas CK							Métricas MOOD						
Módulo	Equipo	WMC	CBO	RFC	LCOM	Promedio	PF	CP	MHF	AHF	MIF	AIF	Promedio
1	1	11%	0%	30%	27%	17%	100%	0%	0%	0%	0%	0%	17%
2	1	9%	45%	45%	58%	39%	0%	0%	0%	0%	0%	0%	0%
3	1	0%	52%	73%	75%	50%	100%	0%	0%	0%	0%	0%	17%
4	2	5%	0%	48%	67%	30%	100%	0%	0%	0%	0%	0%	17%
5	2	1%	57%	75%	84%	54%	0%	0%	0%	0%	0%	100%	17%
6	2	3%	36%	87%	82%	52%	100%	0%	0%	100%	0%	0%	33%
7	2	7%	28%	16%	29%	20%	0%	0%	100%	0%	0%	0%	17%
8	3	2%	25%	71%	51%	37%	0%	0%	0%	0%	0%	0%	0%
9	3	4%	50%	50%	95%	50%	100%	0%	0%	0%	0%	0%	17%
10	3	5%	62%	64%	79%	53%	100%	0%	100%	0%	0%	0%	33%
11	3	0%	40%	67%	33%	35%	0%	0%	0%	0%	0%	0%	0%
12	3	9%	26%	43%	57%	34%	100%	0%	0%	0%	0%	0%	17%
13	4	0%	22%	50%	58%	33%	100%	0%	100%	100%	0%	0%	50%
14	4	0%	35%	78%	87%	50%	0%	0%	0%	0%	100%	0%	17%
15	4	2%	13%	68%	70%	38%	0%	0%	0%	0%	0%	0%	0%
16	4	0%	24%	83%	100%	52%	100%	0%	0%	0%	0%	0%	17%
17	5	19%	33%	58%	80%	48%	0%	0%	0%	0%	0%	0%	0%
18	5	6%	48%	47%	89%	48%	0%	0%	0%	0%	0%	0%	0%
19	5	3%	58%	74%	79%	54%	100%	100%	0%	0%	100%	0%	50%
20	5	0%	13%	56%	81%	38%	100%	100%	0%	0%	0%	0%	33%
21	6	0%	60%	40%	30%	33%	0%	100%	0%	0%	0%	0%	17%
22	6	24%	56%	52%	80%	53%	0%	100%	0%	0%	0%	0%	17%

En la tabla 6 se enlista el número de módulo y equipo, así como el porcentaje de clases cuya medición satisface el valor recomendado para las métricas CK. A partir de estos datos se puede establecer, que en todos los equipos se observa un patrón de resultados similar, sin diferencias considerables, ya que, desde una perspectiva global, al realizar el cálculo por equipo, del promedio de clases cuya medición cumple el criterio de referencia para al menos una de las métricas, se determinó que el equipo con el menor y mayor promedio fue de 35% y 47%, respectivamente (véase figura 1). Es importante mencionar que en la tabla 6 se incluyeron únicamente las cuatro métricas CK que presentan resultados con el mayor porcentaje de clases anómalas.

Adicionalmente, en la tabla 6 se presentan los resultados por módulo y por equipo, para las métricas MOOD, las cuales se aplican a nivel de componente (módulo), en consecuencia, para cada una de las seis métricas, se incluye una columna, que toma el valor de 100% si el resultado de la métrica para dicho módulo satisface el valor recomendado, y 0% en caso contrario. En la última columna se calcula para cada módulo, el promedio de las seis columnas antes referidas. En este caso, nuevamente se observa un patrón de resultados similar en todos los equipos (véase figura 2).

Analizar los componentes anómalos

A partir de los resultados del conjunto de métricas CK presentes en la tabla 4, se estableció que los valores recolectados para las métricas CBO y WMC, presentan el porcentaje más bajo de clases que si cumplen con el valor de referencia, con un 38% y un 5% respectivamente.

En el caso de la métrica CBO, este resultado se obtiene debido a que cada clase anómala presenta un alto acoplamiento con otras clases, es decir, invoca múltiples métodos, o bien usan múltiples atributos de las otras clases. Esto hace que el código de una clase dependa de manera acentuada del código presente en las otras, por esta razón, los cambios en el código fuente de una de ellas, genera una repercusión mayor en el código presente en las clases dependientes. Esta situación, incrementa los costos de mantenimiento. El resultado anómalo para esta métrica indica, además, que la funcionalidad del producto software no se asigna de manera adecuada en las clases Java que la integran.

El resultado para la métrica (WMC) presenta la anomalía más destacable, esta métrica determina la complejidad ciclomática de cada clase, cuyo valor indica el grado de complejidad en la lógica del código

fuente, de esta manera, un valor alto para esta métrica implica un aumento en el grado de dificultad para comprender el algoritmo subyacente en el código de cada clase. En consecuencia, en la muestra de código, existen múltiples clases con una alta complejidad que son difíciles de entender, lo cual nuevamente tiene un impacto en los costos de mantenimiento.

En relación a la categoría de métricas MOOD (véase tabla 5), se establece que el resultado para la métrica CF, excede en 36 unidades porcentuales el valor máximo recomendado. Ésto genera un esfuerzo adicional al realizar las actividades de mantenimiento, ya que el cambio en una clase puede afectar potencialmente a las clases con las que está acoplada. Este resultado es consistente con la medición recolectada para la métrica CBO, que mide el acoplamiento a nivel de clase. Por su parte los resultados de las métricas MIF y AIF, se encuentran por debajo del valor recomendado en 25 y 35 unidades porcentuales, respectivamente. Este hecho, se debe fundamentalmente a que no se aplica de manera adecuada el mecanismo de herencia del POO, es decir, en múltiples clases de la muestra, los atributos y métodos presentes en las clases padre no se reutilizan en las subclases (hijas). Este mecanismo permite reducir la duplicidad del código fuente, y por lo tanto favorecer el reúso y la capacidad de mantenimiento del mismo.

Resultados

Una vez concluido el análisis estático del código fuente, a partir de los valores recolectados en el proceso de medición, se establece el siguiente perfil de fortalezas y debilidades, que adquieren los egresados durante su formación en el programa educativo, para la construcción de código fuente de calidad, enfocada hacia la correcta aplicación de los principios y mecanismos del POO.

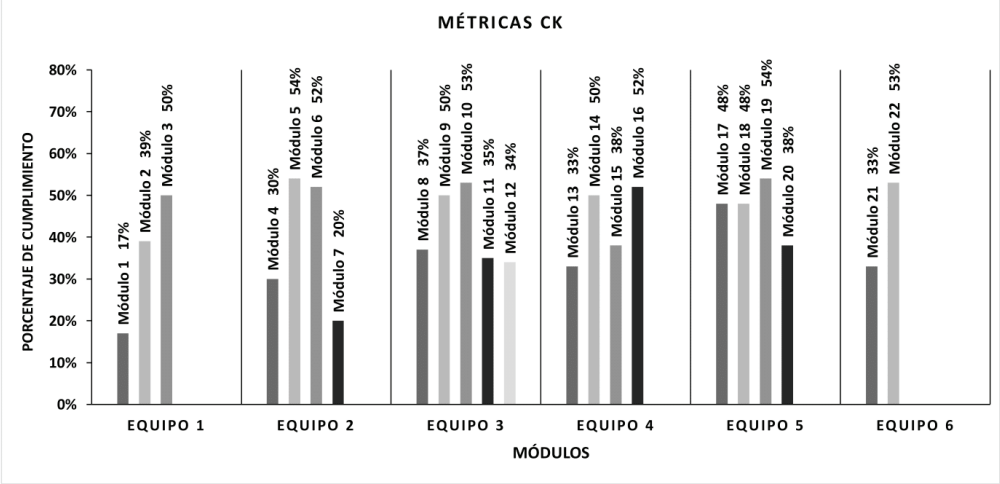

Figura 1. Promedio de cumplimiento de las métricas CK para cada módulo, construido por cada equipo.

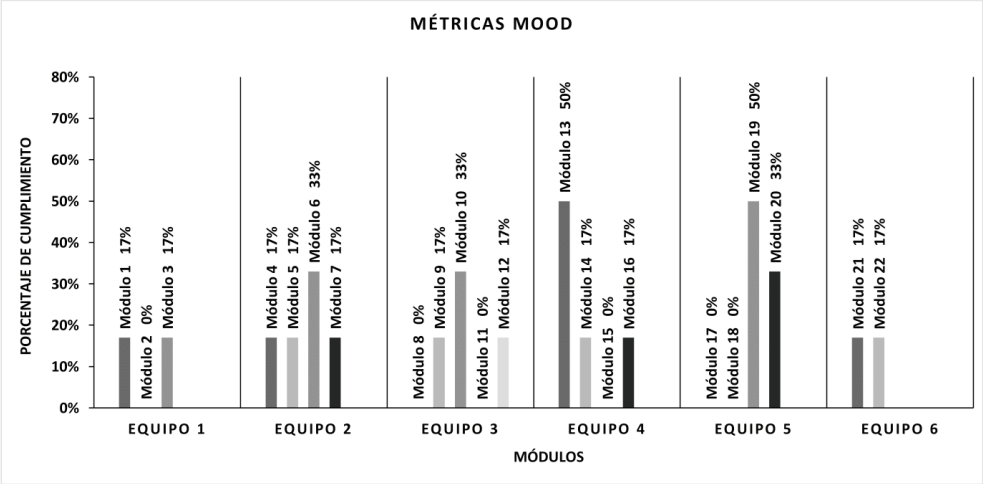

Figura 2. Promedio de cumplimiento del conjunto de las métricas MOOD para cada módulo, construido por cada equipo.

Fortalezas:

- En la jerarquía de clases, se implementa de manera consistente una cantidad adecuada de subclases heredadas (propiedad de herencia).
- Se implementan clases con una cohesión adecuada, es decir, que contienen una funcionalidad correctamente acotada (propiedad de cohesión).
- Se aplica de manera adecuada el mecanismo de ocultamiento de métodos (propiedad de encapsulamiento).
- Se aplica de manera adecuada el mecanismo de ocultamiento de atributos (propiedad de encapsulamiento).

Debilidades:

- Se implementa código con una alta complejidad, cuya lógica de programación es difícil de comprender (propiedad de complejidad).
- Se presenta un alto acoplamiento entre las clases implementadas, que también se observa a nivel de módulo (propiedad de acoplamiento).
- No se utiliza de manera adecuada el mecanismo de herencia de métodos (propiedad de herencia).

Conclusiones

En la industria, el control de la calidad del producto software, es uno de los elementos prioritarios en la gestión de los proyectos de desarrollo. En esta actividad tan relevante, la aplicación de métricas tiene una aportación significativa, ya que, a través de un proceso de medición, es posible establecer un diagnóstico de la calidad del producto software en cada etapa de su ciclo de vida y desde diferentes perspectivas, para de esta manera tomar acciones correctivas o de mejora continua.

Ante este panorama, en la industria se requieren egresados de programas de nivel superior, que posean las competencias necesarias para desarrollar código con los estándares de calidad establecidos. Así pues, en el ámbito del programa educativo, el primer paso para emprender acciones de mejora en su proceso formativo, consiste en el diagnóstico de su situación actual.

En este trabajo se llevó a cabo el análisis estático construido por los egresados para un entorno local y se obtuvo el perfil específico de los egresados para construir código fuente de calidad. Se identificaron las fortalezas y sobre todo las debilidades, que requieren claramente, la mayor atención. Este proceso de medición es replicable para otros programas, siempre que presenten un contexto equivalente al utilizado en el presente estudio.

Los valores recolectados para las métricas CK, ofrecen un diagnóstico sobre el estado de la calidad del código a nivel de clase. Los resultados recolectados para las métricas MOOD, ofrecen un diagnóstico a nivel de módulo, ambos enfoques otorgan un panorama más amplio para obtener el diagnóstico de la calidad del código, durante el análisis estático.

Como trabajo a futuro, a partir de la identificación del perfil de fortalezas y debilidades en la construcción de código fuente de calidad, se requiere diseñar estrategias de mejora, enfocando los esfuerzos principalmente en dos vertientes: el contenido curricular del plan de estudios y los mecanismos pedagógicos aplicados para la enseñanza de la programación bajo el POO.

En el aspecto del contenido curricular es viable poner mayor énfasis en la enseñanza del tema de la calidad del código fuente, así como incluir o en su caso fortalecer la enseñanza de patrones de diseño, que ofrecen soluciones probadas y aceptadas para problemas típicos que se presentan durante la construcción de sistemas software bajo el POO.

Como parte de las estrategias pedagógicas, se plantea abundar en el estudio de la formulación y aplicación de un marco de referencia pedagógico, para la enseñanza de la programación bajo el POO, que ofrezca un conjunto de definiciones, métodos, herramientas y técnicas, con el objetivo de ayudar a consolidar los tópicos del área y aumentar la calidad del código elaborado por los estudiantes.

Finalmente, con el propósito de implementar el proceso de seguimiento de las acciones de mejora, es posible aplicar el análisis estático de manera periódica, y así construir un historial de mediciones, para evaluar el comportamiento de las métricas a lo largo de un periodo.

Bibliografía

Abran A., Moore J. W., Bourque P. & Dupuis R. (2004). SWEBOK: *Guide to the Software Engineering Body of Knowledge v3.0,* IEEE Computer Society.

Alawad, D., Panta, M., Zibran, M., & Islam, M. R. (2018). An Empirical Study of the Relationships between Code Readability and Software Complexity. In *27th International Conference on Software Engineering and Data Engineering* (SEDE) (pp. 1-6).

Antinyan, V., Staron, M., & Sandberg, A. (2017). Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time. *Empirical Software Engineering*, 22(6), 3057-3087.

Breuker, D. M., Derriks, J. & Brunekreef, J. (2011). Measuring static quality of student code. In Proceedings of the 16th annual joint conference on Innovation and technology in computer science education (ITiCSE '11). ACM, New York, NY, USA, 13-17.

Callo Arias, T., Avgeriou, P., America, P., Blom, K., & Bachynskyy, S. (2011). A top-down strategy to reverse architecting execution views for a large and complex software intensive system: An experience report. *Science Of Computer Programming*. 76(12), 1098-1112.

Chidamber, S., & Kemerer, C. (1991). Towards a metrics suite for object oriented design. *ACM SIGPLAN Notices*, (11), 197-211.

Chidamber, S., & Kemerer, C. (1994). A metrics suite for object oriented design. *IEEE Transactions On Software Engineering.* 20(6), 476-493.

Christa, S., Madhusudhan, V., Suma, V., & Rao, J. J. (2017). Software maintenance: From the perspective of effort and cost requirement. In *Proceedings of the International Conference on Data Engineering and Communication Technology* (pp. 759-768). Springer, Singapore.

Dehaghani, S., & Hajrahimi, N. (2013). Which Factors Affect Software Projects Maintenance Cost More?. *Acta Informatica Medica*. 21(1), 63-66.

E Abreu, F. B., & Melo, W. (1996).Evaluating the impact of object-oriented design on software quality. In *Software Metrics Symposium, Proceedings of the 3rd International*, 90-99. IEE.

Erlikh, L. (2000). Leveraging Legacy System Dollars for E-Business. (IEEE) *Information Technology Professional*. 2(3), 17-23.

Favre, L. (2008). Modernizing Software & System Engineering Processes. In *Proceedings ICSENG'08, 19th International Conference*, 442-447. IEEE.

Hashiura, H., Matsuura, S., & Komiya, S. (2010). A tool for diagnosing the quality of Java program and a method for its effective utilization in education. In *Proceedings of the 9th WSEAS International Conference on Applications of Computer Engineering*, 276-282. World Scientific and Engineering Academy and Society (WSEAS).

Hislop, G. W. (2009). Software Engineering Education: Past, Present, and Future. In *Software Engineering: Effective Teaching and Learning Approaches and Practices* (pp. 1-13). IGI Global.

IEEE Std. 1219 (1998). IEEE Standard for Software Maintenance, IEEE Std. 1219-1998. New York, NY.

IEEE Std. 610.2 (1990). IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 610.2-1990.

Kemerer, C., & Slaughter, S. (1999). An empirical Approach to Studying Software Evolution. *IEEE Transactions On Software Engineering*, 25(4), 493-509.

LeBlanc, R. J., Sobel, A., Diaz-Herrera, J. L., & Hilburn, T. B. (2006). Software Engineering 2004: *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*. IEEE Computer Society.

Li, Z., Lu, S., Myagmar, S., & Zhou, Y. (2006). CP-Miner: finding copy-paste and related bugs in large-scale software code. *IEEE Transactions On Software Engineering*, 32(3), 176-192.

Ogheneovo, E. E. (2014). On the Relationship between Software Complexity and Maintenance Costs. *Journal Of Computer And Communications*, 2(14), 1-16.

Pribela, I., Rakić, G. & Budimac, Z. (2012). First Experiences in Using Software Metrics in Automated Assesssment. In *Proceedings of the 15th International Multiconference on Information Society* (*IS*), 250-253.

Radermacher, A., & Walia, G. (2013). Gaps between industry expectations and the abilities of graduates: Systematic literature review findings. In *Proceedings of the 2013 Special Interest Group on Computer Science Education Technical Symposium*. Academic Press.

Romero, R. (2017). *Verificación Automática de Métricas para código en proyectos de software diseñados con el paradigma orientado a objetos* (Tesis de Maestría). Universidad Tecnológica de la Mixteca, Huajuapan de León, Oaxaca, México.

Sommerville, I. (2005). Ingeniería del software. Pearson Educación.

Truong, N., Roe, P., & Bancroft, P. (2004). Static analysis of students' Java programs. In *Proceedings of the Sixth Australasian Conference on Computing Education-Volume*. 30(pp. 317-325). Australian Computer Society, Inc.

Xinogalos, S., & Ivanović, M. (2013). Enhancing Software Quality in Students' Programs. In *Proceedings of 2nd workshop on Software Quality Analysis, Monitoring, Improvement, and Applications* (SQAMIA 2013). CEUR workshop proceedings, 1053, 11-16.