Analizando la Mantenibilidad de Software Desarrollado Durante la Formación Universitaria

Héctor G. Pérez-González, Francisco E. Martínez-Pérez, Sandra E. Nava-Muñoz, Alberto S. Núñez-Varela, Miriam Vázquez Escalante, J. Antonio Flores Saucedo
Facultad de Ingeniería, Universidad Autónoma de San Luis Potosí. Dr. Manuel Nava 8, C.P. 78290, San Luis Potosí. Mexico
hectorgerardo@acm.org, eduardo.perez@uaslp.mx, senavam@uaslp.mx, alberto_snv@hotmail.com, vaem16@gmail.com, flores.s.antonio@gmail.com

***Resumen*— La calidad de los programas de software (SW) es considerada cuando éstos se realizan bajo estrictas metodologías y siguiendo estándares que intentan reducir la probabilidad de presentar defectos. La obtención de métricas de SW es un método útil para minimizar dicha probabilidad. Para evaluar la calidad de un programa de SW, uno de los indicadores más importantes es la mantenibilidad, la cual se puede medir mediante la obtención de un conjunto de métricas. La mantenibilidad es la facilidad con la que un sistema de SW puede ser modificado y es un atributo que afecta de manera crucial al costo de desarrollo del mismo. La literatura muestra estudios sobre mantenibilidad y la alteración de este parámetro cuando los programas evolucionan. Este trabajo muestra los resultados de analizar 315 programas realizados por estudiantes universitarios de carreras en Ingeniería en Computación e Informática y la evolución que la mantenibilidad y otras métricas básicas presentan en función del progreso en la formación académica de sus autores.**

***Palabras clave*— Mantenibilidad, Ingeniería de software, estudio universitario**

I. Introducción

En la actualidad las empresas han estado demandado sistemas de software que sean apegados a la mayoría de los requerimientos de los usuarios y a los estándares de calidad, esto ha incrementado el tamaño de los sistemas de software y su complejidad. Este hecho hace que los sistemas se vuelvan difíciles de mantener y por ende incrementar costos. En este sentido, el glosario de términos del estándar de la IEEE de ingeniería de software define mantenibilidad como[1]:

"La facilidad con la cual un sistema de software o componente puede ser modificado para corregir fallas; mejorar su desempeño u otros atributos; o adaptarse a cambios del ambiente."

La mantenibilidad es un factor altamente significativo en el éxito económico del producto de software. Además es un atributo importante de calidad, pero es difícil de estimar, ya que involucra predicciones acerca de cambios futuros. Por lo que si el modelo de predicciones de mantenibilidad es exacto, entonces el diseño de correcciones podría ser adoptado y ayudar a reducir esfuerzos futuros de mantenibilidad [2]. En esta dirección, es de suma importancia dirigir los esfuerzos en la formación de estudiantes universitarios cuyo rol será el de desarrolladores de software. De forma que al concluir los estudios los ingenieros de software sean capaces de realizar sistemas de software o programas informáticos de calidad bajo estrictas metodologías y siguiendo estándares que intentan minimizar la probabilidad de presentar defectos [3][4]. Estos lineamientos se deben seguir en todas las etapas de desarrollo de software (ingeniería de requerimientos, diseño, programación, pruebas, etc.). Por lo que el objetivo del ingeniero de software es producir programas funcionales (que hagan lo que se supone deben hacer), correctos (sin defectos) y mantenibles (susceptibles de evolucionar ante los cambios manteniendo las dos cualidades anteriores) y que cumplan con los más altos estándares de calidad en programación [3].

No es suficiente que el programa sea funcional de acuerdo con el objetivo por el que fue creado, sino que debe cumplir con una serie de características que lo hagan claro, entendible, flexible y susceptible de ser mantenido. De acuerdo con Rikard y Coleman[5], la mantenibilidad es uno de los atributos más importantes del software.

Obtener el Índice de Mantenibilidad (IM) [6], de un producto de software requiere la obtención de otras métricas básicas las cuales pueden ser correlacionadas tanto con dicho índice como con el progreso del entrenamiento referido.

En este artículo se presenta un estudio en el cual se analiza la evolución de la mantenibilidad de los programas de software durante la formación de ingenieros de software. Además se muestran otras métricas consideradas para el análisis de 315 programas desarrollados por los estudiantes comparando el progreso de sus habilidades de programación durante su entrenamiento académico en el transcurso de su carrera profesional.

En la sección 2 se establecen los fundamentos de medición de software y se muestran los resultados de la revisión bibliográfica acerca de métricas, poniendo especial énfasis en la noción de mantenibilidad y como ésta ha sido estudiada de manera relativa a la evolución de los productos de software.

En las secciones 3 y 4 se establecen los objetivos y la metodología de esta investigación. Aquí se describe el origen de los programas analizados, las fases en que se divide la formación académica de los desarrolladores y las métricas a ser aplicadas.

Finalmente en las secciones 5 y 6 se muestran los resultados y su análisis, así como las conclusiones, aportaciones, lecciones aprendidas y una propuesta de trabajo futuro.

II. Revisión de la Literatura: Métricas de Software, Mantenibilidad y Estudios Comparativos

La medición de software es el proceso mediante el cual se puede obtener un valor numérico a partir de un proceso o producto de software [6]. Comparando estos valores con un conjunto de estándares (definidos por cada individuo, grupo de desarrollo de software, organización, etc.), es posible obtener conclusiones acerca de la calidad del software. La medición de software se logra generalmente mediante métricas.

Las métricas de software se clasifican en métricas de proceso y de producto [6]. Este trabajo se enfoca

exclusivamente en las métricas de producto, en particular, aquellas métricas relacionadas con el código fuente y cómo obtenerlas.

Las métricas de producto son mediciones del producto de software en cualquier estado de su desarrollo; desde los requerimientos hasta el sistema instalado y son ampliamente usadas por la industria del software en sus procesos de medición[6].

Existe una gran variedad de métricas que permiten la medición de las características del software, sin embargo como lo establece Aquilino[7], se deben definir atributos claros que permitan establecer un sistema adecuado de medidas para evaluar un producto final de software. Por otra parte, Briand y Morasca[8], afirman que las métricas son un buen medio para entender, monitorizar, controlar, predecir y probar el desarrollo del software y los proyectos de mantenimiento. Además, tal y como señala Pfleeger [9] las métricas pueden ser utilizadas por los profesionales e investigadores para tomar las mejores decisiones en relación a la eventual asignación de tareas a los equipos de diseñadores, programadores o personal de soporte.

Por otro lado, la norma internacional ISO / IEC 9126 para la calidad de productos de software define tres perspectivas sobre la calidad de estos productos: calidad interna, calidad externa y calidad de uso[10]. Estas perspectivas corresponden a tres fases distintas del ciclo de vida de un producto de software.

La calidad interna se refiere a la calidad del producto la cual puede percibirse por la observación del mismo sin considerar su funcionamiento. Para determinar la calidad interna del producto se utilizan técnicas de análisis estático tales como la obtención de métricas de código fuente.

La calidad externa del producto puede ser observada en la fase de pruebas mediante el análisis dinámico de su comportamiento.

La calidad de uso del producto es la percibida por sus usuarios cuando el software está en funcionamiento.

Para el estudio de la calidad interna y externa, la norma ISO / IEC 9126 considera seis características principales y veinte sub-características que en conjunto conforman la noción de calidad. Una de las seis características principales es la mantenibilidad. Para evaluar estas características se utilizan métricas de producto de software.

Una métrica de software es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado[1] .

La mantenibilidad se subdivide en: analizabilidad, variabilidad, estabilidad y la capacidad de prueba[1].

La analizabilidad es la facilidad para localizar elementos y posibles fallas dentro de un código. La variabilidad es el esfuerzo necesario para realizar cambios en el software, la estabilidad es la probabilidad de que los cambios en el software den lugar a fallos y la capacidad de prueba es el esfuerzo requerido para probar cambios en el software.

De acuerdo con [5], la mantenibilidad se calcula en función de otras métricas.

Dyshlevyy[11] realizó un estudio en el que define un conjunto mínimo de mediciones para hacer un análisis de mantenibilidad.

De acuerdo con Rikard y Coleman [5][12] mantenibilidad es uno de los más importantes atributos del software.

Rosenberg establece que la calidad del código se conforma de cinco atributos: Eficiencia, complejidad, entendibilidad, reusabilidad y mantenibilidad[13].

Wilson a partir de un estudio con 75 desarrolladores internacionales establece la importancia que éstos dan a los siguientes factores de calidad: Reusabilidad, flexibilidad, eficiencia, capacidad de prueba, mantenibilidad, usabilidad, confiabilidad y correctés.

Diferentes autores han realizado estudios comparativos de atributos de mantenibilidad caracterizándolos en diferentes productos o en los mismos productos tras haber realizado modificaciones en ellos. Como por ejemplo Coleman y Ash de Hewlett-Packard, reportan cómo el análisis automático de mantenibilidad se puede utilizar para guiar la toma de decisiones en el desarrollo de software[12]. Rikard Land de la Universidad de Malardalen investigó cómo la mantenibilidad de una pieza de software cambia a medida que pasa el tiempo y experimenta mantenimiento o evolución. Esto lo dedujo mediante la obtención de métricas de sistemas de la industria[5].

Los estudios de Riaz et al., 2009 [14] sugieren que existe evidencia de que los estudios de mantenibilidad permiten predecir las características que tendrán los productos de software. Ohlsson et al., 1999 [15] utilizaron métricas de mantenibilidad para concluir que existe un decrecimiento de la calidad del código en sistemas legados tras sucesivas versiones del mismo. Ganpati et al., 2012 [16] sostiene que la mantenibilidad de cualquier sistema de software se cuantifica en términos del Índice de Mantenibilidad (IM), en su estudio analizaron empíricamente, cincuenta versiones de sistemas de software abierto escritos en PHP producidas en un periodo de siete años. Ellos también obtuvieron las relaciones entre otras métricas de software (Líneas de código, Complejidad ciclomática y Volumen Healstead). Los resultados revelan que hay un decremento en el Índice de Mantenibilidad conforme se generan nuevas versiones del software.

De la revisión de la literatura se puede deducir que la mantenibilidad es una noción crucial en la caracterización de productos de software ya que puede ser útil en la toma de decisiones para las etapas de evolución de los mismos. La mantenibilidad puede ser identificada a partir de la obtención de métricas de software tales como el Índice de Mantenibilidad (IM).

Se han realizado estudios para determinar la relación que existe entre la mantenibilidad de diferentes versiones de productos de software. No se encuentran evidencias de estudios que relacionen la mantenibilidad de productos de software realizados por un mismo autor, sea éste desarrollador profesional o estudiante en formación.

De lo anterior se plantea la siguiente pregunta de investigación: ¿Existe correlación entre la mantenibilidad del software y el progreso en la etapa de formación académica de sus autores?

III. Objetivo del estudio

Las carreras de Ingeniería en Informática e Ingeniería en Computación de la Universidad Autónoma de San Luis Potosí (UASLP), desde su creación, se han caracterizado por su arduo trabajo a lo largo de la formación del estudiante, llevando durante su estancia, proyectos de desarrollo tecnológico. Estos proyectos no tienen un valor sumativo en la evaluación del curso pero son un requisito para tener derecho a ella; su entrega incluye documentación y código fuente en medios de almacenamiento. Este trabajo analiza una muestra de estos proyectos tomada desde el año 1995 hasta el año 2007, para obtener la mantenibilidad de los programas y analizar la relación que ésta tiene con métricas básicas de código y con parámetros relacionados con el progreso de los autores como estudiantes de dichas carreras. Cabe hacer notar que durante este periodo se realizaron dos importantes reestructuraciones en los planes de estudio de las carreras y en el personal

académico asociado. En estos cambios, se enfatizó las mejoras en el aprendizaje de la ingeniería de software.

IV. Metodología

Se realizó un estudio en el cual, inicialmente se tomó una muestra de 405 proyectos del total de los proporcionados por el Área de Computación e Informática de la UASLP. La selección se llevó como se expone a continuación.

Se realizó una clasificación de los proyectos tomando como referencia la documentación de cada proyecto. Esta clasificación fue realizada según la materia a la cual pertenecían los proyectos.

Posteriormente se consideraron las materias en donde se realizan proyectos con una relativamente alta exigencia en programación para dividirlas en tres niveles como se muestra en la Figura 1. En un primer nivel se incluyeron las materias de: Lenguajes de programación A, Estructuras de Datos y Algoritmos, y Tecnología Orientada a Objetos; éstas por ser materias en las cuales se imparten los principios básicos y los diferentes paradigmas (tipos) de programación (estructurado y orientado a objetos). Los proyectos desarrollados al cursar estas materias son básicamente simulaciones simples y videojuegos. El segundo nivel lo conformaron las materias de: Administración Informática, Sistemas de Información I, Sistemas Operativos I y Sistemas Operativos II; por ser materias en las que se da libertad al alumno de seguir el paradigma que considere sea el más apropiado. Los proyectos desarrollados al cursar estas materias son de tipo empresarial, simulaciones más complejas y de software de aplicación. El tercer nivel se conformó por las materias de: Ingeniería de software II (materia en la que se imparten los principios de calidad de software para llegar a elaborar un código funcional y de alta calidad), Sistemas Expertos, Teleproceso, Programación de Sistemas I y Programación de Sistemas II, tomando en cuenta que éstas son materias más complejas y de los últimos semestres. Los proyectos desarrollados al cursar estas materias se relacionan con programación de sistemas de software de base, administración de sistemas y procesamiento complejo de datos.

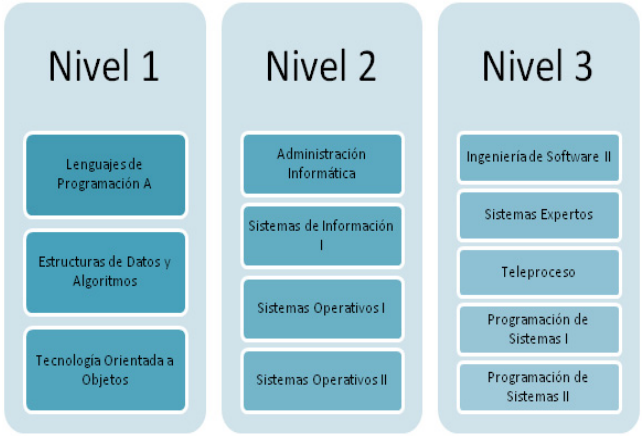

Fig. 1. Kilo líneas de código (KLOC) por nivel.

Tomando esta clasificación, se recopiló el código fuente de cada proyecto, el cual se encontraba almacenado en discos de 5 1⁄4'' (Diskettes), una mayoría en discos de 3 1⁄2'' (Diskettes) y solo unos pocos en CD (Disco Compacto); algunos no contaban con código o los archivos que los contenían estaban dañados. En algunos más, el medio de almacenamiento no se encontraba en buenas condiciones. De esta última selección se logró obtener un total de 315 programas de software. Se obtuvieron los siguientes datos de los proyectos: materia, semestre y autor. Cuando se tuvo reunida dicha información se procedió a evaluar los códigos haciendo uso del software "Krakatau Professional for C/C++"[17][18]. Se utilizaron las siguientes métricas propuestas por Ganpati et al. 2012 [16]:

LOC (o KLOC): Total de líneas (o kilo líneas de código), sin contar comentarios o líneas en blanco.

COM-RAT: Radio de comentarios. Relación entre líneas de comentarios con respecto al total de código.

V: Máximo volumen por método (*Halstead Program Volume*). Este parámetro depende de la cantidad de operadores (N) y operandos (n) utilizados en el programa y se relaciona con la cantidad de bits necesarios para almacenarlo. Se define como V=Nlog2n, donde N=N1+N2, siendo N1 el número de ocurrencias de operadores y N2 el número de ocurrencias de operandos. Y n=n1+n2, siendo n1 el número de operadores diferentes y n2 el número de operandos diferentes.

E (EFFORT): Esfuerzo o dificultad. Se puede interpretar como una medida del trabajo requerido para escribir y para comprender un software ya desarrollado. Se obtiene de E=V*D, donde V es el volumen y D (dificultad)= (n1 / 2) * (N2/n2), (n1= número de operadores únicos, n2 = número de operandos únicos y N2=número total de operandos). Los operadores pueden ser +, -, =, etc. Los operandos son los valores (constantes o variables) utilizados en las operaciones.

TABLA I. Métricas de programas analizados

	Promedios por materia					
Descripción	*LOC*	*COM RAT*	*V(G)*	*V*	*E*	*SEIMI*
Estructura de datos y Algoritmos	1353.14	0.03806	4.5709	1050.35	773312.82	-86.99
Lenguajes de programación A	1445.92	0.03469	15.819	1008.80	1560498.4	-126
Tecnología Orientada a Objetos	2208.36	0.06752	2.9694	379.24	804625.66	19.34
Administración Informática	1805.5	0.046	8.155	945.715	418876.65	-138.73
Sistemas de Información I	920	0.0305	3.29	688.49	126606.86	-101.21
Sistemas Operativos I	882	0.0502	2.4557	339.923	201309.17	-32.6
Sistemas Operativos II	951.461	0.074	4.6805	644.920	289724.44	-75.82
Ingeniería de Software II	5363.96	0.21437	1.7464	124.569	231647.18	60.59
Programación de Sistemas I	3575.31	0.07205	7.1730	900.130	2193576.8	-163.33
Programación de Sistemas II	2758.46	0.0879	5.2587	661.144	765722.65	-99.42
Sistemas Expertos	1507.5	0.066	3.0725	330.569	212835.47	-6.57
Teleproceso	1560.14	0.1728	3.34485	248.499	135625.40	-25.13

TABLA II. Resultados Agrupados por niveles

Nivel	*LOC*	*COM RAT*	*V(G)*	*V*	*E*	*SEIMI*
1	1676.46	0.05	7.73	809.05	1038783.9	-64.52
2	1314.18	0.05	4.45	664.41	334869.69	-87.09
3	2954.03	0.12	4.11	450.91	703116.77	-46.77

V (G): Complejidad ciclomática [13]. La complejidad de un programa se mide mediante el número ciclomático (v) de su grafo de flujo (G*): v(G)=e–n+2*, donde *e* es el número de arcos y n el número de nodos de G. Es útil para la predicción de módulos propensos a errores, esfuerzo de mantenimiento y reutilización.

SEIMI: Índice de Mantenibilidad de un proyecto (SEI por sus siglas en inglés de Carnegie Mellon University).. Este mide la facilidad de mantenimiento correctivo y evolutivo[13] y viene dado por:

$$SEIMI = 171 - 5.2 \bullet \ln(avgV) - 0.23 \bullet avgV(G) - 16.2 \bullet \ln(avgLOC) + 50 \bullet \sin(\text{sqrt}(2.4 \bullet perCM))$$

Donde: avgV = Volumen promedio por método, avgV(G)= Complejidad ciclomática promedio, avgLOC= Líneas de

código medio por método y perCM: = % de líneas de comentarios.

V. Resultados y Análisis

Se analizaron 315 programas que en conjunto estaban constituidos por 573,220 líneas de código. Se obtuvo el Índice de Mantenibilidad (SEIMI) con el objetivo de analizar su relación con el progreso en la formación académica de sus autores. Así mismo y con el objetivo de correlacionar diferentes métricas básicas con dicho Índice de Mantenibilidad se obtuvieron las métricas siguientes: Número de líneas de código (KLOC), Radio de comentarios (CPM-RAT), Volumen Halstead (V), Esfuerzo de entendibilidad (E) y Complejidad ciclomática (V(G))

Una vez aplicada la evaluación a todos los programas fuente, se obtuvieron datos de las métricas descritas.

La Tabla I muestra los datos obtenidos de la aplicación de las seis métricas seleccionadas. Las tres materias iniciales conforman el nivel 1, las siguientes 4 el nivel 2 y el resto constituyen el nivel 3, las columnas muestran las métricas obtenidas.

La Tabla II muestra la información obtenida agrupada en los tres niveles propuestos que conforman las carreras para cada una de las métricas. Por ejemplo, el promedio de líneas de código (LOC) utilizadas en los proyectos conforme el estudiante avanza en su carrera; inicia con un promedio de 1,676 LOCs en el primer tercio y concluye con un promedio de 2,954 LOCs en cada uno de sus proyectos al final de las carreras (Ver Figura 2).

La Figura 3 muestra la métrica COM-RAT, el radio de comentarios; los estándares recomiendan un rango de 20 al 35 % de líneas de comentarios con respecto del total del programa [19]. Se aprecia que el porcentaje mejora a partir del año 2004.

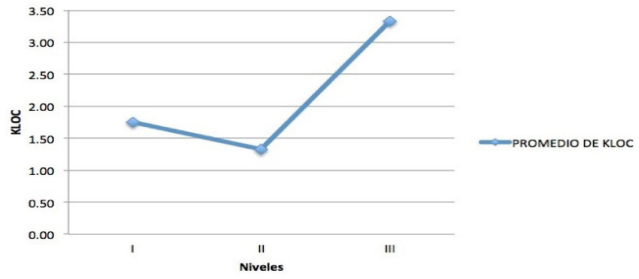

Fig. 2. Kilo líneas de código (KLOC) por nivel.

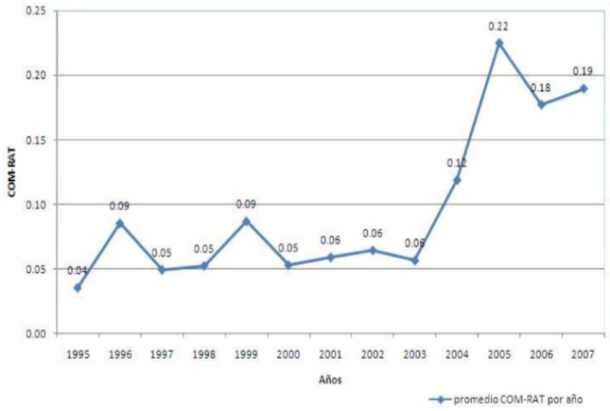

Fig. 3. Radio de comentarios en programas por año.

La Figura 4 muestra que el volumen (V) de los programas disminuye del nivel I al nivel II y de éste al nivel III. Esto significa que el conjunto de operandos y operadores utilizados en los programas disminuye. Esta variación es previsible ya que si bien los programas en semestres más avanzados presentan mayor complejidad en su lógica, estos reducen su necesidad de demasiadas variables y operaciones.

La Tabla II ilustra que el esfuerzo (E) para el desarrollo de los programas al final de las carreras es significativamente mayor al de la etapa intermedia pero menor que el de la primera fase. Estos valores se incrementan no obstante a que el volumen disminuye.

La complejidad ciclomática (V(G)) por método o función representa el grado de facilidad o dificultad para entender modificar o corregir un método o función dentro de un programa. Esta complejidad aumenta con el número de caminos que el programa pueda seguir. Se recomienda mantener éste valor menor a 6[13] . La Figura 5 muestra que en los últimos dos tercios de las carreras los programas son de mayor calidad dado que la complejidad ciclomática promedio de cada método o función no excede el valor máximo recomendado.

Los resultados de este trabajo fueron también clasificados de acuerdo al año en que se desarrollaron los proyectos; estos se agruparon del año 1995 al 2000, del 2001 al 2002 y del 2003 al 2007. Lo anterior debido a que en los años 1999 y 2003 las carreras experimentaron actualizaciones tanto de los planes y programas de estudio como del personal docente. Estos cambios dieron inicio a periodos de análisis de impacto de los cambios emprendidos.

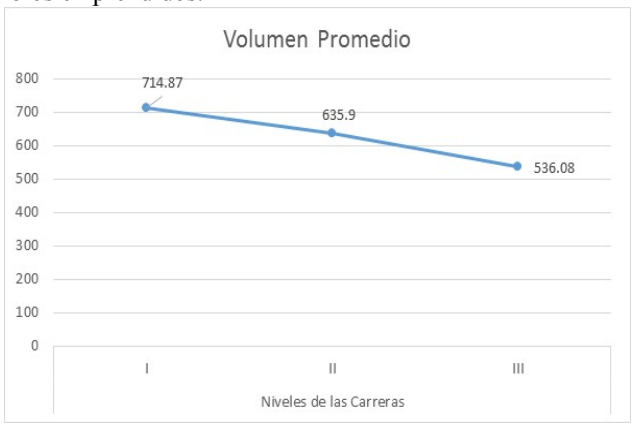

Fig. 4. Volumen de los programas por nivel.

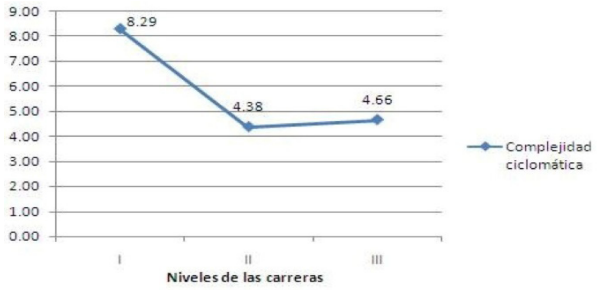

Fig. 5. Complejidad ciclomática por método.

La Figura 6 muestra que el grupo de proyectos correspondientes al periodo 1995-2000 muestra un nivel medio de SEIMI al inicio de la carrera, este es mayor en el segundo nivel y menor al final de la misma. Las modificaciones en las materias del final de las carreras efectuadas en 1999 se refleja en la sección 2001-2002 de la gráfica; sin embargo, el índice sufre una reducción importante en las materias del nivel II. Tras las modificaciones del 2003 (principalmente en las materias del nivel I) el SEIMI observa un nivel máximo en el nivel inicial y una leve recuperación hacia el final de las carreras.

A través del análisis de la fórmula propuesta por el SEI se observa que el Índice de Mantenibilidad es directamente

proporcional al porcentaje de líneas de comentarios en el programa e inversamente proporcional al volumen del código, a su complejidad ciclomática y a su tamaño en líneas de código. Se concluye que aunque el tamaño de los programas crece del inicio al final de las carreras académicas, el Volumen de ellos disminuye. El parámetro que contribuye a elevar el índice de mantenibilidad es el crecimiento del radio de comentarios en los programas. Aun así la complejidad de todos los programas y su radio de comentarios insuficiente genera valores de mantenibilidad desfavorables (negativos) en todos los productos de software.

En síntesis, en los últimos años se ha observado un avance muy importante para el nivel I, para el cual los alumnos no cuentan con conocimiento de conceptos que coadyuven a la mantenibilidad del software. Con base en lo anterior podemos percibir que las mejores prácticas de programación no son aplicadas en los cursos del tercio medio de las carreras y su mantenibilidad se reduce por el aumento de la complejidad de los mismos. Se destaca que aunque la complejidad de los programas desarrollados al final de las carreras es aún mayor, el índice de mantenibilidad y en consecuencia la calidad de los productos recupera su nivel.

VI. Conclusiones y Trabajo Futuro

Con base en la información podemos observar los siguientes hechos:

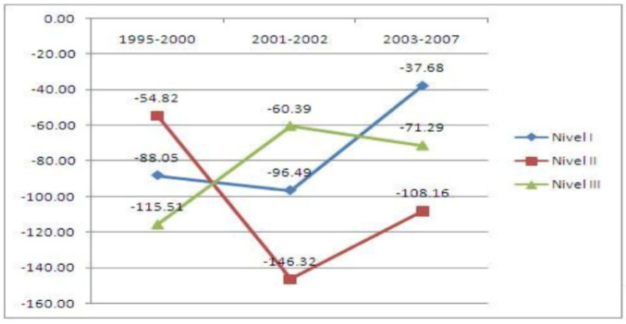

Fig. 6. Promedios SEIMI por periodos y niveles.

A lo largo de la carrera los estudiantes pasan por tres etapas. En la primera, ellos escriben programas relativamente pequeños en líneas de código, pero con un volumen y una complejidad ciclomática relativamente altos. La segunda etapa se caracteriza por la disminución de estas tres métricas (tamaño, volumen y complejidad ciclomática). Durante la tercera etapa muestra un aumento considerable del tamaño de los programas; sin embargo, el volumen sigue disminuyendo y la complejidad prácticamente se mantiene igual. En cuanto al radio de comentarios se observa en la tabla 2 que durante las primeras dos etapas éste es bajo, en la tercera éste se duplica.

La figura 6 muestra que en los proyectos de las generaciones o periodos 1995-2000 el índice de mantenibilidad aumenta de la etapa 1 a la 2 y disminuye drásticamente de la etapa 2 a la 3. En los proyectos de los periodos 2001-2002 el índice de mantenibilidad disminuye gravemente de la etapa 1 a la 2 y aumenta considerablemente de la etapa 2 a la 3. Finalmente en los proyectos de los periodos 2003-2007 el índice de mantenibilidad disminuye drásticamente de la etapa 1 a la 2 y aumenta de la etapa 2 a la 3.

Con respecto a las métricas básicas obtenidas en relación al año o generación de los alumnos, la figura 2 muestra una sostenida mejora de los programas en cuanto a la documentación del código. Es importante considerar que las carreras (donde los autores de estos programas estaban enrolados) experimentaron en los años 1999 y 2003 actualizaciones tanto de los planes y programas de estudio como del personal docente. Estas conllevaron una agresiva política de mejoras en la enseñanza de la ingeniería de software.

Con lo anterior podemos concluir que existe evidencia empírica de la relación directa entre la formación académica de los estudiantes y las características de sus programas. Esto se deriva del hecho de que una vez que las estructuras académicas son modificadas, los índices de mantenibilidad se ven sumamente afectados a pesar de que las métricas básicas de tamaño, volumen y complejidad permanecen similares.

A. Contribución, lecciones aprendidas y Trabajo Futuro

Las métricas básicas de código de software y en particular el Índice de Mantenibilidad fueron analizadas sobre un conjunto de 315 programas creados por alumnos de carreras de Computación e Informática. Esto para un periodo de 12 años. Durante este periodo se realizaron modificaciones estructurales a los programas académicos y se observó la incidencia de estos cambios en las métricas analizadas.

Previo al estudio, se realizó una investigación de la literatura referente al estudio comparativo de métricas en programas que experimentan evolución. La literatura muestra que en general conforme aumenta el tamaño (KLOC) de los programas, el Índice de Mantenibilidad decrece, lo cual es un hecho intuitivamente esperado.

Se destaca que como resultado de este estudio, se observó que la mantenibilidad de los programas escritos por estudiantes durante su formación es directamente proporcional al tamaño de los mismos.

La principal lección aprendida consiste en que los procesos de enseñanza-aprendizaje pueden tener la suficiente influencia para incrementar la mantenibilidad del software del estudiante incluso conforme se incrementa el tamaño de dichos productos.

Adicionalmente se deduce de la tabla 2, que una vez que los planes de estudio y la planta docente se alinean con los objetivos de generar aprendizaje de conceptos de mantenibilidad en los estudiantes se genera una correlación entre dicha métrica y el tamaño del programa. Esto se observa dado que los programas realizados en la primera etapa muestran un tamaño relativamente cercano al promedio de la muestra; este tamaño decrece a sus niveles mínimos en la segunda etapa para crecer a sus niveles máximos en la etapa final. Lo anterior sucede de manera muy similar en los índices de mantenibilidad en los programas de las tres etapas.

Como trabajo futuro se sugiere la posibilidad de ampliar el trabajo con estudiantes de más recientes generaciones. Así mismo se propone dar seguimiento a los estudiantes ya convertidos en egresados que realizan programas en la industria o en estudios de posgrado.

Referencias

[1] [1] I. 610. 12199. Std, “IEEE Standard Glossary of Software Engineering Terminology,” pp. 1–84, 1990.

[2] [2] J. Saraiva, “A roadmap for software maintainability measurement,” in Software Engineering (ICSE), 2013 35th International Conference on, 2013, pp. 1453–1455.

[3] [3] R. E. Fairley, Ingeniería de Software. Mc Graw Hill, 1990.

[4] [4] P. P. Texel, “Measuring Software Development Status : Do We Really Know Where We Are ?,” in Proceedings of the IEEE SoutheastCon 2015, 2015.

[5] [5] R. Land, “Measurements of Software Maintainability,” Proc. ARTES Grad. Student Conf., pp. 1–7, 2002.

[6] [6] I. Sommerville, Ingeniería de Software, 8a ed. Pearson Education, 2006.

[7] [7] J. A. A and A. Aquilino, Necesidad de Sistemas Formales de Métricas para Proyectos de Software OO. Universidad de Oviedo, 1997.

[8] [8] L. C. Briand and S. Morasca, "Property-based software engineering measurement," Softw. Eng. IEEE Trans., vol. 22, no. 1, pp. 68–86, 1996.

[9] [9] S. L. Pfleeger, "Assessing Software Mesasurement," IEEE Softw., vol. March/Apri, pp. 25–26, 1997.

[10] [10] I. Iso, "Software Engineering-Product Quality-Part 1: Quality Model," Geneva, Switz. Int. Organ. Stand., 2001.

[11] [11] O. P. Dyshlevyy and K. V. Shaposhnikova, "TOOL OF DEFINING MINIMAL SCOPE OF MEASUREMENTS FOR MAINTAINABILITY ANALYSIS," vol. 4, pp. 6–8, 2012.

[12] [12] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using metrics to evaluate software system maintainability," IEEE Comput., vol. 27, no. 8, pp. 44–49, 1994.

[13] [13] L. Rosenberg and L. Hyatt, "Software quality metrics for object-oriented environments," Crosstalk Journal, April, pp. 1–7, 1997.

[14] [14] M. Riaz, E. Mendes, and E. Tempero, "A Systematic Review of Software Maintainability Prediction and Metrics.pdf," in Third International Symposium on Empirical Software Engineering and Measurement, 2009, pp. 367–377.

[15] [15] M. C. Ohlsson, a. Von Mayrhauser, B. McGuire, and C. Wohlin, "Code decay analysis of legacy software through successive releases," in 1999 IEEE Aerospace Conference. Proceedings (Cat. No.99TH8403), 1999, vol. 5, pp. 68–81.

[16] [16] A. Ganpati and A. Kalia, "Maintainability Index over Multiple Releases : A Case Study PHP Open Source Software," vol. 1, no. 6, pp. 6–8, 2012.

[17] [17] M. Lanza and R. Marinescu, Object-Oriented Metrics in Practice. Springer-Verlag Berlin Heidelberg, 2006.

[18] [18] "KRAKATAU metrics," 2015. [Online]. Available: http://www.powersoftware.com/kp/.

[19] [19] J. C. Munson, Software Engineering Measurement. CRC Press, 2003.

Héctor G. Pérez González obtuvo el grado de Maestro en Ciencias Computacionales en la Universidad Nacional Autónoma de México in 1993 y el de Doctor en Ciencias Computacionales en la Universidad de Colorado en 2003. Sus áreas de interés son la Ingeniería de Software y la Interacción Humano Computadora.

Francisco E. Martínez-Pérez obtuvo el Título de Ingeniero en Computación en la Facultad de Ingeniería de la Universidad Autónoma de San Luis Potosí (UASLP) en 2001, el grado de Maestro en Ciencias Computacionales por la UASLP in 2005 y el grado de Doctor en Ciencias en la Universidad Autónoma de Baja California en 2012. Actualmente es administrador del laboratorio UDICEI. Sus áreas de interés son la Interacción Humano Computadora, Ingeniería de Software, Cómputo ubicuo y procesamiento de imágenes.

Sandra E. Nava-Muñoz obtuvo el Título de Ingeniero en Computación en la Facultad de Ingeniería de la Universidad Autónoma de San Luis Potosí (UASLP) en 2001, el grado de Maestro en Ciencias Computacionales por la UASLP in 2005 y el grado de Doctor en Ciencias en la Universidad Autónoma de Baja California en 2013. Sus áreas de interés son la Interacción Humano Computadora, Ingeniería de Software y Cómputo ubicuo.

Alberto Nuñez-Varela obtuvo el Título de Ingeniero en Computación en la Universidad Autónoma de San Luis Potosí (UASLP) en 2005, el grado de Maestro en Ingeniería en Computación en el Centro de Investigación y Estudios de Posgrado en la UASLP en 2011. Actualmente es estudiante del programa de Doctorado en Ciencias Computacionales y profesor en la Universidad Autónoma de San Luis Potosí. Su área de interés es la Ingeniería de Software, en particular, Calidad de Software y Métricas de software.

Miriam Vázquez-Escalante obtuvo el Título de Ingeniera en Informática en la Facultad de Ingeniería de la Universidad Autónoma de San Luis Potosí (UASLP) en 2012. Actualmente es responsable de la Escuela de Control en el ICMJO. Su área de interés es la Ingeniería de Software.

José Antonio Flores-Saucedo obtuvo el Título de Ingeniero en Computación en la Facultad de Ingeniería de la Universidad Autónoma de San Luis Potosí (UASLP) en 2015. Actualmente se desempeña como desarrollador Web y sus áreas de interés son la Ingeniería de Software y Tecnologías Web.