Universidad Autónoma de Querétaro

Facultad de Informática

Método de análisis y medición automática de código fuente para establecer el grado de cumplimiento de las propiedades del Paradigma Orientado a Objetos

Tesis

Que como parte de los requisitos para obtener el grado de

Maestro en Sistemas Computacionales

Presenta

I.S. Raquel Mondragón Huerta

Santiago de Querétaro, Qro., Noviembre 2018

Universidad Autónoma de Querétaro
Facultad de Informática
Maestría en Sistemas Computacionales

Método de análisis y medición automática de código fuente para establecer el grado de cumplimiento de las propiedades del Paradigma Orientado a Objetos

TESIS
Que como parte de los requisitos para obtener el grado de
Maestro en Sistemas Computacionales

Presenta:
I.S. Raquel Mondragón Huerta

Dirigido por:
M.I.S.D. Carlos Alberto Olmos Trejo

SINODALES

M.I.S.D. Carlos Alberto Olmos Trejo Presidente	Firma
Dr. Jorge Adalberto Torres Jiménez Secretario	Firma
Dra. Rosa María Romero González Vocal	Firma
M.I.S.D. Juan Salvador Hernández Valerio Suplente	Firma
M.I.S.D. Jesús Armando Rincones Suplente	Firma

M.I.S.D. Juan Salvador Hernández Valerio
Director de la Facultad

Dra. Ma. Guadalupe Flavia Loarca Piña
Director de Investigación y Posgrado

Centro Universitario
Santiago de Querétaro, Qro.
Diciembre / 2018
México

RESUMEN

El objetivo de la investigación fue diseñar un método de análisis y medición del código fuente que permitiera establecer el grado de cumplimiento de las propiedades de programación del Paradigma Orientado a Objetos mediante la metodología basada en el diseño de Hevner, March, Park & Ram (2004). El análisis de los perfiles propuestos por la Association for Computing Machinery (ACM) en colaboración con el Committee for Computing Education in Community Colleges (CCE) permitió determinar las habilidades y competencias de programación en el área de Ingeniería de Software. La vinculación de las habilidades y competencias relacionados a los conocimientos en programación del Paradigma Orientado a Objetos tiene como base el modelo del Institute of Electrical and Electronics Engineers (IEEE). Finalmente, la evaluación del grado del cumplimiento de las propiedades vinculadas a las habilidades y competencias de programación en el Paradigma Orientado a Objetos tiene base en la utilización de las métricas más importantes para medir el software.

(**Palabras clave**: evaluación, métricas, programación, POO, habilidades, competencias, software)

SUMMARY

The aim of this work is at designing a method of source code analysis and measurement to establish the level of compliance with the programming properties of the Object-Oriented Paradigm, using the methodology based on the design of Hevner, March, Park & Ram (2004). The profiles analysis proposed by the Association for Computing Machinery (ACM) in collaboration with the Committee for Computing Education in Community Colleges (CCE) enabled to determine the programming skills and competences in the area of Software Engineering. Linking skills and competences related to programming knowledge of the Object-Oriented Paradigm is based on the model of the Institute of Electrical and Electronics Engineers (IEEE). Finally, the assessment of the level of compliance with those properties linked to programming skills and competences in the Object-Oriented Paradigm is based on the use of the most important metrics to measure software.

(**Key words**: assessment, metrics, programming, OOP, skills, competences, software)

A mi familia

Como una muestra de mi cariño y agradecimiento, por todo el amor y apoyo brindado y porque hoy veo llegar a su fin una meta más en mi vida.

Con admiración y respeto

Raquel Mondragón Huerta

AGRADECIMIENTOS

En primer lugar, me gustaría dar las gracias a los directores del proyecto el Mtro. Carlos Alberto Olmos Trejo y al Dr. Jorge Torres Jiménez por la idea inicial y por todo el apoyo en cada uno de los objetivos propuestos, a la Dra. Rosa María Romero González por la revisión final del presente documento y los constantes consejos para la culminación del mismo, me gustaría agradecer al Mtro. Juan Salvador Hernández Valerio y al Mtro. Jesús Armando Rincones por su apoyo académico y personal.

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACyT) y a la Universidad Autónoma de Querétaro por el apoyo y patrocinio depositado en el desarrollo de este proyecto.

A Pablo, por su constante apoyo y ayuda, no solo en el tiempo que trabajamos juntos en el desarrollo de nuestros respectivos trabajos, también por haber estado en mi vida, por compartir conmigo cada experiencia de vida.

Por último, quiero dar las gracias a mis amigos con los que he compartido todos estos años y especialmente agradezco a mi familia, mis padres, mis hermanos y mi novio, cuyo esfuerzo y trabajo diario han sido indispensables para conseguir todos mis retos.

TABLA DE CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

1. INTRODUCCIÓN

El proceso de aprendizaje de la programación no es una tarea fácil y requiere del desarrollo de habilidades del pensamiento, y un cambio de paradigma en la forma de resolver problemas. Rosson y Carroll (1990) establecen que comprender el Paradigma Orientado a Objetos significa entender que son los objetos y como los mensajes se transfieren entre ellos con el fin de realizar tareas específicas. Los estudiantes durante el proceso de aprendizaje se enfrentan a diferentes dificultades al tratar de entender conceptos abstractos de la Programación Orientada a Objetos relacionando equivalencias de esos conceptos en la vida real (Yan, 2009). Por otro lado, el papel que juegan los docentes es complicado y determinante en el proceso de aprendizaje. Ellos son los encargados de poner en práctica los métodos de enseñanza y de evaluar la evolución del aprendizaje de los alumnos (Rodríguez, 2014).

Las evaluaciones y mediciones del aprendizaje se han realizado de manera tradicional mediante la asignación de notas numéricas a los estudiantes. Sin embargo, existen dudas y malas prácticas sobre qué es lo que se debe de evaluar y que se debe de obtener de ello, contribuyendo a hacer que sea más importante la obtención de una buena nota que la obtención de conocimientos significativos. Esto genera que no se cuenten con los indicadores necesarios para medir la calidad del aprendizaje (Aguilar, 2009). Dentro del seguimiento y evaluación de las habilidades y competencias de la Programación Orientada a Objetos la pregunta es en qué grado los profesores son capaces de analizar un código para evaluar la utilización de las propiedades del Paradigma Orientado a Objetos visibles a través de un

método de evaluación de caja negra, el cual se muestra en la Figura 1.1, donde a partir de entradas especificas se generan salidas esperadas. Este tipo de evaluación limita la visibilidad de la utilización de las propiedades del paradigma.

Figura 1.1 Evaluación de caja negra. Fuente; Elaboración propia con base en Aguilar (2009)

En la Figura 1.2 se muestra un método de caja blanca permitiría tener una mejor visión de la solución y de que conceptos el alumno está utilizando para ello.

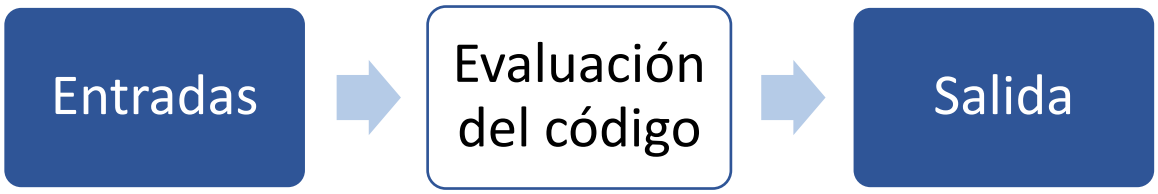

Figura 1.2 Evaluación de caja blanca. Fuente: Elaboración propia con base en Aguilar (2009)

1.1 Justificación del estudio

El proceso de aprendizaje de la programación no es una tarea fácil y requiere del desarrollo de habilidades de pensamiento, y un cambio de paradigma en la forma de resolver problemas. Rosson y Carroll (1990) establecen que comprender el paradigma orientado a objetos significa entender que son los objetos y como los mensajes se transfieren entre ellos con el fin de realizar tareas específicas (Rodríguez, 2014).

Los estudiantes durante el proceso de aprendizaje se enfrentan a diferentes dificultades al tratar de entender conceptos abstractos de la Programación Orientada a Objetos relacionando equivalencias de esos conceptos en la vida real (Yan, 2009). Por otro lado, el papel que juegan los docentes es complicado y determinante en el proceso de aprendizaje. Ellos son los encargados de poner en práctica los métodos de enseñanza y de evaluar la evolución del aprendizaje de los alumnos.

Las evaluaciones y mediciones del aprendizaje se han realizado de manera tradicional mediante la asignación de notas numéricas a los estudiantes. Sin embargo, existen dudas y malas prácticas sobre qué es lo que se debe de evaluar y que se debe de obtener de ello, contribuyendo a hacer que sea más importante la obtención de una buena nota que la obtención de conocimientos significativos. Esto genera que no se cuenten con los indicadores necesarios para medir la calidad del aprendizaje (Aguilar, 2009).

Dentro del seguimiento y evaluación de las habilidades y competencias de la programación orientada a objetos la pregunta es en qué grado los profesores son capaces de analizar un código para evaluar la utilización de las propiedades del paradigma orientado a objetos visibles a través de un método de evaluación de caja negra donde a partir de entradas especificas se generan salidas esperadas. Este tipo de evaluación limita la visibilidad de la utilización de las propiedades del paradigma. Un método de caja blanca permitiría tener una mejor visión de la solución y de que conceptos el alumno está utilizando para ello.

El proceso de aprendizaje de la programación no es una tarea fácil y requiere del desarrollo de habilidades de pensamiento, y un cambio de paradigma en la forma de resolver problemas. Rosson y Carroll (1990) establecen que comprender el paradigma orientado a objetos significa entender que son los objetos y como los mensajes se transfieren entre ellos con el fin de realizar tareas específicas (Rodríguez, 2014).

Los estudiantes durante el proceso de aprendizaje se enfrentan a diferentes dificultades al tratar de entender conceptos abstractos de la Programación Orientada a Objetos relacionando equivalencias de esos conceptos en la vida real (Yan, 2009). Por otro lado, el papel que juegan los docentes es complicado y determinante en el proceso de aprendizaje. Ellos son los encargados de poner en práctica los métodos de enseñanza y de evaluar la evolución del aprendizaje de los alumnos.

Las evaluaciones y mediciones del aprendizaje se han realizado de manera tradicional mediante la asignación de notas numéricas a los estudiantes. Sin

embargo, existen dudas y malas prácticas sobre qué es lo que se debe de evaluar y que se debe de obtener de ello, contribuyendo a hacer que sea más importante la obtención de una buena nota que la obtención de conocimientos significativos. Esto genera que no se cuenten con los indicadores necesarios para medir la calidad del aprendizaje (Aguilar, 2009).

Dentro del seguimiento y evaluación de las habilidades y competencias de la programación orientada a objetos la pregunta es en qué grado los profesores son capaces de analizar un código para evaluar la utilización de las propiedades del paradigma orientado a objetos visibles a través de un método de evaluación de caja negra donde a partir de entradas especificas se generan salidas esperadas. Este tipo de evaluación limita la visibilidad de la utilización de las propiedades del paradigma. Un método de caja blanca permitiría tener una mejor visión de la solución y de que conceptos el alumno está utilizando para ello.

1.2 Objetivo General

Es diseñar un método que permita evaluar, de manera efectiva, el grado de cumplimiento de las habilidades y competencias de la programación del Paradigma Orientado a Objetos basado en el código fuente.

1.3 Objetivos Específicos

- Construir un modelo de habilidades y competencias requeridas en la formación del Paradigma Orientado a Objetos.
- Elaborar un mapa de asociaciones entre propiedades del Paradigma Orientado a Objetos con las habilidades y competencias en la formación de este mismo paradigma; que permita determinar el aprendizaje de los estudiantes.
- Desarrollar un modelo de métricas para evaluar la presencia o ausencia de las propiedades basado en un código fuente.

1.4 Supuesto de la Investigación

Si se utiliza un método de evaluación de código fuente haciendo uso de métricas de software entonces obtendremos el grado de cumplimiento de las habilidades y competencias de la programación del Paradigma Orientado a Objetos.

1.5 Diseño Metodológico de la Investigación

La metodología elegida para el desarrollo de la investigación es la basada en el diseño. Esta metodología es apropiada para investigaciones donde se pretende trabajar para ampliar los límites de las capacidades humanas por medio de la creación de artefactos innovadores (Hevner, March, Park & Ram, 2004). La elaboración de esta investigación debe tener el rigor necesario para sustentar las

contribuciones o desarrollos futuros. En la figura 1.3 se muestra el marco de trabajo de dicha metodología, la cual se basa en la relevancia del entorno y el rigor de la base del conocimiento para brindar la formalidad que requiere un trabajo de investigación.

Figura 1.3 Marco de trabajo de la metodología de investigación basada en diseño. Fuente: Elaboración propia con base en Havner et al. (2004)

Siguiendo los objetivos de la investigación, se consideró apropiado utilizar la metodología de Investigación Basada en el Diseño (IBD). Esta metodología implementa un diseño instructivo que se elabora, implementa y somete a un rigor de investigación, de allí que los estudios se desarrollen para la introducción de

nuevos temas, nuevas herramientas o nuevos modelos del contexto de aprendizaje (Confrey, 2006).

La IBD se concentra en el diseño y desarrollo de innovaciones educativas, considerando posibles artefactos como por ejemplo software como el centro de esas innovaciones, contribuyendo así a una mejor comprensión del entorno y las condiciones del aprendizaje (Bell, 2004).

Rinaudo, Chiecher y Donolo (2010) mencionan cuatro características de los estudios de diseño:

a. Ubicar el problema de investigación en el contexto natural en el que ocurren los fenómenos de estudio.
b. El propósito principal es producir cambios específicos en el contexto.
c. Centrarse en los enfoques sistémicos, es decir en los estudios que tratan a las variables como independientes y transaccionales.
d. Mantener el carácter cíclico e iterativo de los diseños.

Adoptar esta metodología en el campo educativo permite generar conocimiento que contribuya a mejorar la calidad de las practicas instructivas en diferentes niveles, contextos y áreas disciplinarias (Rinaudo y Donolo, 2010).

En base a la metodología de investigación basada en el diseño se describen las siguientes fases para el desarrollo de la investigación:

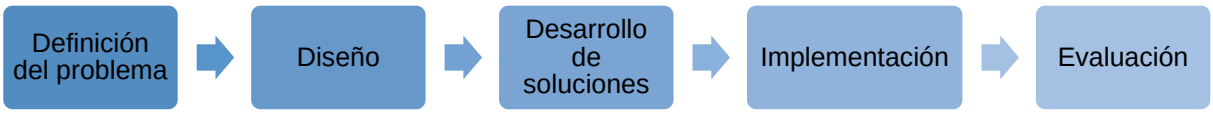

Figura 1.4 Fases para el desarrollo de la investigación. Fuente: Elaboración propia

1. Definición del problema: La fase número uno tiene como propósito identificar la problemática emergente de la interacción de las personas, las organizaciones y la tecnología.
2. Diseño: La fase número dos consiste en la investigación de las bases del conocimiento, corresponde al estado del arte y tiene como propósito indagar en todos los cimientos teóricos y metodológicos que existen para que permita encontrar la solución a las problemáticas antes planteadas.
3. Desarrollo de las soluciones: La fase número tres consiste en la integración de la solución, es decir en la identificación del entorno correcto y las base de conocimiento suficiente para la generación de la solución.
4. Implementación: La fase número cuatro tiene el propósito de aplicar las soluciones y someterse a una constante evaluación para refinar los nuevos conocimientos.
5. Evaluación: La fase número cinco permite corroborar las aportaciones y documentar el método propuesto para cumplir con los objetivos establecidos.

1.6 Problema de Investigación

La formación en la Programación Orientada a Objetos no solo es codificar, es la integración de grupos de habilidades para el desarrollo de software. Desarrollar software es un proceso complejo que se ha definido a lo largo de los años, la cual involucra distintas actividades que requieren de las siguientes habilidades y competencias específicas (McConnell, 2004). La evaluación efectiva en el proceso de formación en la Programación Orientada a Objetos no se cumple mediante la técnica de evaluación de caja negra ya que no se permite una correcta valoración de las propiedades del paradigma. Un método de caja blanca permitiría tener una mejor visión de la solución y de que conceptos el alumno está utilizando para ello.

El método de caja blanca implica una gran inversión de tiempo de los docentes, y estas revisiones por el número de alumnos podrían verse afectadas en la calidad debido a la complejidad de analizar diferentes códigos fuentes, por ello se plantea que este método se aplique de manera automática en base al código fuente de cada uno de los estudiantes. Utilizar un método de caja blanca automatizado del código fuente de cada uno de los estudiantes nos permitiría mejorar la asertividad de la evaluación del aprendizaje, dar seguimiento a las características individuales de cada estudiante, proporcionar una retroalimentación personalizada y automática, impulsar el aprendizaje de los estudiantes, apoyar en la impartición del conocimiento a los profesores y permitir a las Instituciones Educativas llevar un control de perfiles.

Por lo tanto, el objetivo general de esta investigación es diseñar un método que permita evaluar, de manera efectiva, el grado de cumplimiento de las habilidades y competencias de la programación del Paradigma Orientado a Objetos

basado en el código fuente, y la hipótesis de investigación “Los componentes de software son capaces de evaluar de manera automática el aprendizaje de la programación orientada a objetos.”

La formación en la Programación Orientada a Objetos no solo es codificar, es la integración de grupos de habilidades para el desarrollo de software. Desarrollar software es un proceso complejo que se ha definido a lo largo de los años, el cual involucra distintas actividades que requieren de habilidades y competencias específicas (McConnell, 2004). La evaluación efectiva en el proceso de formación en la Programación Orientada a Objetos no se cumple mediante la técnica de evaluación de caja negra ya que no se permite una correcta valoración de las propiedades del paradigma. Un método de caja blanca permitiría tener una mejor visión de la solución y de que conceptos el alumno está utilizando para ello.

El método de caja blanca implica una gran inversión de tiempo de los docentes, y estas revisiones por el número de alumnos podrían verse afectadas en la calidad debido a la complejidad de analizar diferentes códigos fuentes, por ello se plantea que este método se aplique de manera automática en base al código fuente de cada uno de los estudiantes. Utilizar un método de caja blanca automatizado del código fuente de cada uno de los estudiantes nos permitiría mejorar la asertividad de la evaluación del aprendizaje, dar seguimiento a las características individuales de cada estudiante, proporcionar una retroalimentación personalizada y automática, impulsar el aprendizaje de los estudiantes, apoyar en la impartición del conocimiento a los profesores y permitir a las Instituciones Educativas llevar un control de perfiles.

Por lo tanto, el objetivo general de esta investigación es diseñar un método que permita evaluar, de manera efectiva, el grado de cumplimiento de las habilidades y competencias de la programación del Paradigma Orientado a Objetos basado en el código fuente, y la hipótesis de investigación “Los componentes de software son capaces de evaluar de manera automática el aprendizaje de la programación orientada a objetos.”

2. ESTADO DEL ARTE

2.1 La importancia del Paradigma Orientado a Objetos

En el mundo de la programación existen muchos lenguajes mediante los cuales se pretenden cumplir las necesidades de desarrollo que la industria y formación del software requiere. En la Figura 2.1 se muestra el listado de los lenguajes más populares según el sitio codeval.com durante el 2016, donde Python ocupaba el primer lugar.

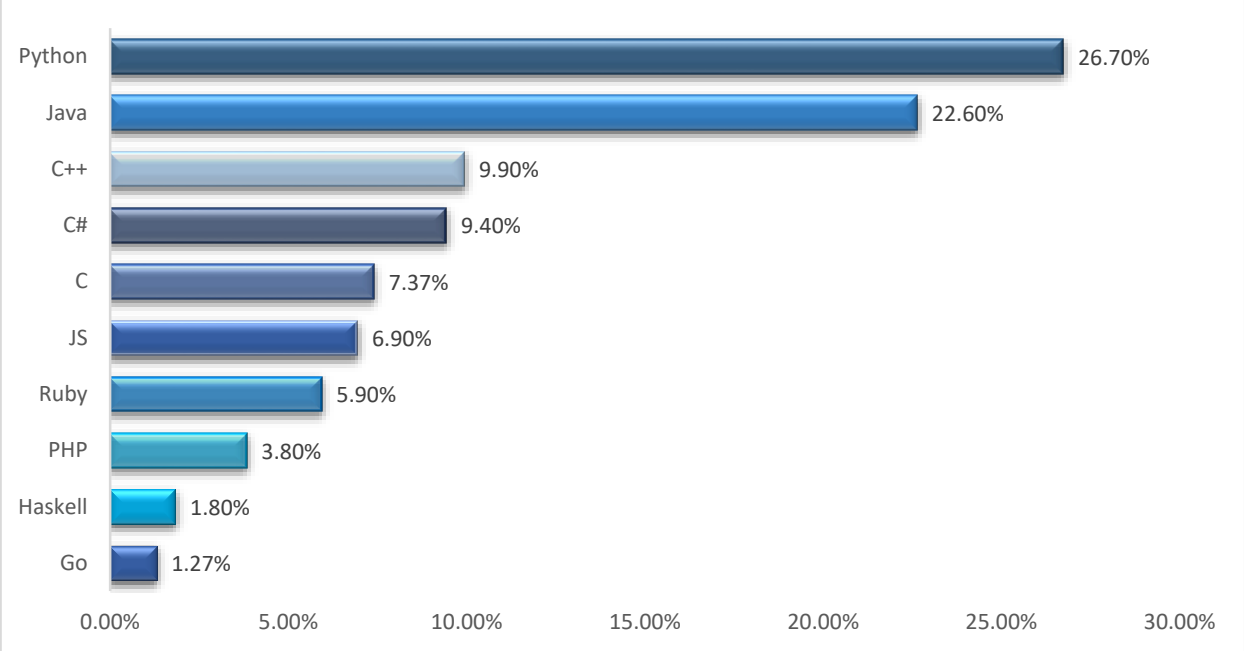

Figura 2.1 Lenguajes de programación más populares 2016. Fuente: Elaboración propia con base en Codeval (2016)

Este lenguaje desde hace ya varios años ha tomado mucha fuerza, ya que cuenta con características que facilitan el desarrollo de software. Python es actualmente el lenguaje de programación más utilizado para enseñar a codificar desplazando a java, ya que es uno de los lenguajes más fáciles para que los principiantes aprendan, y no solo dentro de la ingeniería de software sino que también es adaptable a ambientes con propósitos científicos como en temas de automatización, procesamiento de imágenes o análisis numérico. Como este lenguaje de programación es multiplataforma, los estudiantes pueden comenzar con el desarrollo de habilidades simples y avanzar hasta adquirir habilidades más complejas de desarrollo haciendo uso de la programación del Paradigma Orientado a Objetos (Vidya y Toby, 2014).

La Programación Orientada a Objetos (POO) es actualmente un paradigma de programación importante dentro de la Ingeniería de Software. (Gadja, 2011). Cada vez son más los requerimientos que se demandan de un software y con ello crece la complejidad durante el desarrollo. Crear software bajo un paradigma estructurado demanda el 67% del costo total durante el mantenimiento (Sommerville, 1992), así también el diseño se vuelve complejo y por la naturaleza de este paradigma el trabajo por grupos se complica. Booch (1996) menciona que la programación orientada a objetos debe ser utilizada para el desarrollo de software de “dimensión industrial”, ya que algunas características como reutilización de código, polimorfismo, herencia y encapsulación, por mencionar algunas, permiten manejar la complejidad durante el proceso de desarrollo y diseño (Booch, 1996).

2.2 Habilidades de formación en el paradigma orientado a objetos

Formar para la construcción de software en un paradigma orientado a objetos es más que solo codificar. Desarrollar software es un proceso complejo que se ha definido a lo largo de los años, el cual involucra distintas actividades que requieren de grupos de habilidades y competencias específicas (McConnell, 2004). Estas actividades son: definición del problema, desarrollo de requerimientos, planeación de la construcción, diseño de alto nivel: arquitectura de software, diseño detallado, codificación y depuración, pruebas de unidad, pruebas de integración, pruebas de sistema, y mantenimiento correctivo

Para formar en un paradigma la industria de software toma diferentes referentes importantes dentro de la Ingeniería de Software para sustentar los conocimientos que los profesionistas en el área deben de desarrollar. Uno de estos referentes es la IEEE (Institute of Electrical and Electronics Engineers) que propone un modelo de competencias a desarrollar en la Ingeniería de Software.

Las habilidades de construcción de software son utilizadas para desarrollar y diseñar la arquitectura de un sistema basándose en los requerimientos, durante el proceso se requiere una descripción de cómo el software se integra de componentes. Los componentes se especifican en un nivel de detalle que permite su construcción. Esta clasificación de habilidades también incluye competencias relacionadas con procesos y técnicas para diseño de software de calidad, análisis y evaluación. Todo este proceso lo estratifica en cuatro grupos generales de habilidades, las cuales son: Fundamentales, Estrategias y métodos, diseño

arquitectónico y Calidad, análisis y evaluación (IEEE, 2014) como se observa en la Figura 2.2.

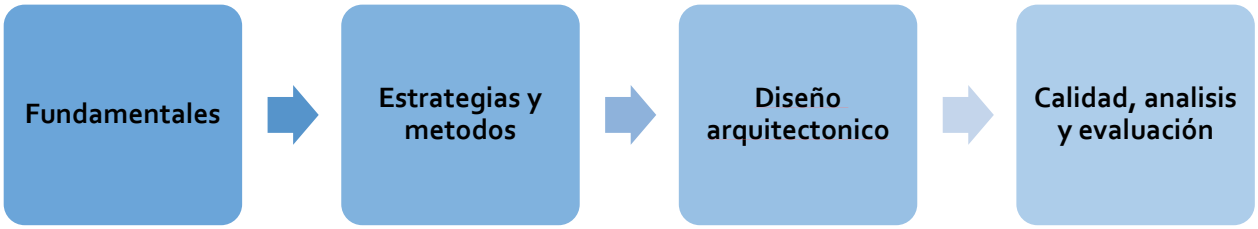

Figura 2.2 Grupos generales de habilidades según la IEEE. Fuente: Elaboración propia con base en IEEE (2014)

Así como la programación Orientada a Objetos es ampliamente utilizada en la industria del software, también las Universidades forman en este paradigma. Las instituciones educativas asumen la responsabilidad de formar a sus estudiantes con las habilidades y competencias requeridas, todo esto a través de la utilización de métodos de enseñanza y aprendizaje que permitan lograrlo de la mejor manera (Moreno, 2004).

Varios métodos y modelos de enseñanza del conocimiento se han desarrollado a lo largo del tiempo poniéndolos en práctica para determinar cuál de ellos es el más efectivo. Sin embargo sus resultados varían según el tipo de conocimiento que se desea transmitir y a quienes se les enseña. Entre una de las teorías de enseñanza más utilizada se encuentra el cognitivismo, la cual busca desarrollar habilidades a través de la percepción y la memoria en donde el razonamiento y la autoevaluación juegan un papel primordial; por otro lado está el

conductismo, el cual intenta construir el conocimiento a través de los sentidos (Albert, 2005).

Para el proceso de enseñanza-aprendizaje de la programación orientada a objetos existen varias técnicas y una de las más conocidas y practicadas es la de Kolb, quien desarrolló un teoría experimental del aprendizaje a la que llamó “Ciclo de aprendizaje” (Kolb, 1984). Esta teoría es un ciclo que se clasifica en cuatro etapas, las cuales se describen en la Tabla 2.1.

Tabla 2.1

Etapas del ciclo de aprendizaje de Kolb.

Etapa	Descripción
a) Experimentación Activa	En esta etapa los estudiantes realizan un ejercicio de programación, identifican los recursos específicos de programación que necesitan por ejemplo, las constantes, las variables o las estructuras de control, y establecen criterios para encontrar una solución al ejercicio propuesto.
b) Acción	Los estudiantes escriben y compilan el programa que da solución al problema propuesto.
c) Observación reflexiva	En esta etapa los estudiantes pueden observar los resultados de la compilación: errores de compilación o los resultados de la ejecución. En este momento los estudiantes reúnen información y reflexionan sobre los resultados.
d) Conceptualización	Finalmente, los estudiantes generan las conclusiones del análisis de la información disponible para aprender de su experiencia. Los estudiantes utilizan los criterios previamente establecidos y las pruebas para determinar el grado de logro de su objetivo. Si no se logró

	satisfactoriamente los objetivos los estudiantes usarán sus conclusiones para iniciar de nuevo el ciclo en la etapa de experimentación activa donde planifican una estrategia revisando el conocimiento adquirido anteriormente.

Fuente: Elaboración propia con base en Kolb (1984).

En la Figura 2.3 se presenta el ciclo de Kolb, el cual se sustenta en generar conocimiento con base en la experiencia. (Rodríguez, 2014).

Figura 2.3 Ciclo del aprendizaje de Kolb. Fuente: Elaboración propia con base en Rodríguez (2014)

2.3 Evaluación del aprendizaje

Un aspecto importante durante el proceso de aprendizaje es la evaluación del mismo. La evaluación permite medir el aprendizaje y determinar si se realiza de manera correcta y se obtienen los resultados esperados. Para ello establecer determinadas métricas que comprueben la efectividad con la que el conocimiento es transferido correctamente a los estudiantes. Para dar una idea más clara de los resultados la mayoría de los sistemas actuales de evaluación manejan de manera numérica para representar las habilidades o competencias que los estudiantes poseen en diferentes áreas del conocimiento (Moreno, 2004).

Los resultados de las evaluaciones son utilizadas para obtener información y tomar decisiones sobre el proceso educativo que llevan los estudiantes, los métodos de enseñanza que aplican los profesores y los programas educativos en general, por ello son tan importantes las evaluaciones para las Instituciones Educativas (Santos, 2009).

La necesidad de evaluar a los estudiantes durante su formación en la programación orientada a objetos genera alguna problemática acerca de la manera en cómo se valora actualmente a este tipo de cursos. La mayoría de los profesores, siguiendo el ciclo de aprendizaje de Kolb, en donde los profesores generan problemas que dejan a los estudiantes para su resolución. Los estudiantes analizan las entradas requeridas y las salidas deseadas para generar una solución la cual codifican y ejecutan. Los formadores solo verifican que esas salidas sean correctas aplicando un método de evaluación de caja negra donde el proceso de solución no

pasa por un proceso de análisis para identificar las propiedades utilizadas del paradigma orientado a objetos dentro de su solución.

Un aspecto importante durante el proceso de aprendizaje es la evaluación del mismo. La evaluación permite medir el aprendizaje y determinar si este se está realizando de manera correcta y se tienen los resultados esperados. Para ello se establecen ciertas métricas que comprueben que efectivamente el conocimiento es transferido correctamente a los estudiantes. Para dar una idea más clara de los resultados la mayoría de los sistemas actuales de evaluación se maneja de manera numérica para representar las habilidades o competencias que los estudiantes poseen en diferentes áreas del conocimiento (Moreno, 2004).

2.4 Métodos de evaluación del aprendizaje

Existen distintas formas de verificar que efectivamente el aprendizaje se ha llevado a cabo de la forma correcta, y para ello se describen algunas de las formas más comunes empleadas en la actualidad y en el ámbito académico y se contrastan con los métodos actuales que presenta la Facultad de Informática de la Universidad Autónoma de Querétaro.

En la actualidad en el contexto académico se pretende asignar de forma cuantitativa una evaluación al aprendizaje académico de un alumno en competencias cualitativas y es por ello existen diversas propuestas que pretenden representar los conocimientos, habilidades y competencias que los alumnos efectivamente poseen (Bordas, 2001).

Una de las formas más básicas de calificar que una habilidad es realizable por un alumno es decir si es apto o no para realizar dicho trabajo, dentro del contexto académico se puede etiquetar como “apto” o "no apto", para ello se verifican que las competencias existan o no existan llevando a cabo actividades que verifiquen prácticamente o teóricamente ese sustento. Liu (2009) dice que cuando se habla de un curso específico el resultado se puede etiquetar como “aprobado", es decir “apto", o no “aprobado” o "no apto". Sin embargo para un dominio de un tema amplio como son cursos especializados dentro de una universidad se ha optado dada la complejidad de los cursos por evaluar las competencias a una mayor granularidad y que permitan dar indicadores más amplios que una simple etiqueta que indique si es o no apto para realizar un trabajo, es por ello que se ha representado numéricamente la competencia total del desarrollo de una actividad o curso del que se esté hablando para poder decir si las aptitudes están presentes.

El sistema numérico o representación numérica en una escala finita de las habilidades de un alumno se puede representar de forma continua o discreta siempre y cuando esté representada en un intervalo determinado, por ejemplo para representar la ausencia de atributos del aprendizaje se empieza definiendo una escala que empiece con el número cero, y se definen niveles secuenciados hasta llegar a una nota final máxima que se puede obtener como lo puede ser el número 5 (Moreno, 2004).

El ejemplo mencionado es parte del sistema de calificaciones usado en varios países de la unión europea y nórdica. Por ejemplo en Finlandia en la Universidad de Helsinki cuenta con un sistema mixto con materias que indican si un alumno ha

aprobado o no un curso (apto o no apto) mientras que otras indican en una escala numérica discreta del cero al 5 (cero: no apto, del 1 al 5 apto) las habilidades adquiridas en un curso (Valdivia, 2008).

Dentro del intervalo seleccionado se definen atributos cualitativos que representan el aprendizaje adquirido, es decir una nota con valor de 1 significa que el alumno está aprobado en la materia mientras que una nota con 4 o 5 indica que lo hizo de manera sobresaliente. De la misma manera que Finlandia evalúa a sus alumnos con una escala del cero al 5, la educación impartida por el gobierno mexicano emplea una escala numérica discreta similar del cero al 10, donde del cero hasta el número 5 se indica un progreso no satisfactorio o aprobado de las materias mientras que del 6 al 10 lo hacen de manera satisfactoria pero dan más idea de la cantidad de conocimiento que pudieron adquirir sin dar mayor información que un número global.

Avila (2007) plantea que otra forma similar al método discreto por intervalo numérico es el continuo, donde se permite asignar una calificación con fracciones decimales. En la libre cátedra, los profesores que deben de reportar calificaciones de forma numérica discreta y en su proceso evaluativo terminan con números fraccionarios deciden si hacen algún tipo de redondeo lo que de alguna forma les da mayor flexibilidad en cuanto al manejo de los puntos.

Una manera más de representación del aprendizaje en un tema es por porcentaje. En esta forma se calcula el número de competencias logradas entre el número de competencias totales impartidas o transmitidas por el profesor, de tal forma que al final se pueda obtener una representación discreta o continua entre

cero y cien por ciento o la unidad (1.0) como máximo. Siguiendo con este esquema se puede dejar hasta el nivel discreto mencionando el número de competencias logradas dejando de lado o mencionado el total de las posibles competencias adquiridas (Silvias, 2006).

Pasando de la forma de evaluación individual a la colectiva también es posible decir que si un grupo determinado de alumnos cumplen o no con ciertas competencias y para ello la forma de evaluar puede cambiar dado que los actores involucrados son diferentes. Se puede optar por evaluar al grupo como una ente donde se les califica a todos por igual dependiendo de los resultados que obtengan todos, sin embargo esto no nos da mucha idea de las competencias individuales y esto es parte de la problemática que presentan varias universidades en el mundo para poder dar una retroalimentación personalizada dada la complejidad que ahora presenta y asegurar que individualmente los alumnos tienen competencias básicas necesarias para cumplir con un perfil deseado (Núñez, 2005).

Existen distintas formas de evaluación por equipos en donde se espera que los integrantes sean capaces en conjunto de solucionar cierto problema; dependiendo de la dinámica se pretende centrar el aprendizaje en la resolución del problema mismo, aunque globalmente se evalúa la solución del contenido dada la complejidad que implica darle seguimiento personalizado a cada integrante del equipo a menos que se indique explícitamente en el trabajo dicha aportación, sin embargo, es probable que queden áreas sin explorar o desarrollar para cada integrante y eso generara deficiencias individuales si el objetivo es aprender dichos contenidos más que resolver el problema en equipo (Moreno, 2004).

Una forma de pasar las habilidades o unidades cualitativas a cuantitativas es asignar mediante una rúbrica en intervalos cuantificables definir en cada intervalo un número o porcentaje que represente del total o del límite las habilidades que se esperan medir. Se puede dar una breve descripción para verificar que efectivamente un alumno evaluado cae en alguno de los intervalos con claridad. Al final se suman los porcentajes de atributos presentes en el individuo y se le da una calificación representativa de dicha actividad. Una variación de esto es que cada intervalo defina cualitativamente las habilidades del individuo para poderlo asignar a una casilla donde estén presentes la mayoría o todas sus habilidades descritas pero implica menor granularidad en la evaluación y el modelo es comparable al de una regla, donde en cada unidad se describe el progreso que debe de seguir el actor. El método en general se puede aplicar al nivel de atomicidad deseado donde para cada sección se aplica el mismo proceso y al final los elementos más pequeños son a los que se les asigna su puntuación sobre el total o el porcentaje relevante.

Podemos observar dentro de las formas de evaluación colaborativas que se pueden elaborar coevaluaciones involucrando a los miembros del equipo que estén resolviendo la problemática. Los miembros dan su opinión acerca del desempeño de sus colegas por haber trabajado cercanamente. Este modelo está limitado a la percepción que los involucrados puedan tener de los otros miembros y no necesariamente reflejan consistentemente el aprendizaje logrado por cada uno de ellos, inclusivamente cuando la pregunta o cuestión no es la de evaluar a sus colegas, sino evaluarse a sí mismo, ya que el punto de partida es la persona misma. Es posible decir individualmente si se considera que las habilidades se han

incrementado para cierta área, pero aun así no se cuenta con un marco externo que dictamine que efectivamente las aptitudes cubren cierto nivel mínimo esperado.

2.5 La evaluación automática

Las habilidades de programación son claves en el área de la Ingeniería de Software y la Informática, pero al paso de los años esta relevancia va tomando fuerza en otras disciplinas no solo para las Ciencias Computacional, sino también en áreas como la Ingeniería, las Ciencias Sociales, las Matemáticas y la Economía. El aprendizaje y la enseñanza de la programación es un factor crítico para obtener un título en Ciencias de la Computación y también en el desarrollo de investigación de otras disciplinas, por ello una correcta evaluación y una apropiada retroalimentación de la misma en importante en el desarrollo del aprendizaje (Fangohr & O'Brien, 2015).

Una de las habilidades que definen a un profesionista como Ingeniero de software es la programación. La programación es una tarea creativa donde dadas las restricciones del lenguaje de programación a utilizar, es la elección del programador qué estructura de datos es la más apropiada a usar, qué flujo de control implementar, qué paradigma de programación implementar, como nombrar variables y funciones, cómo documentar el código, y cómo estructurar el código para resolver el problema en unidades más pequeñas que potencialmente podrían ser reutilizadas (Robins, Rountree & Rountree, 2003).

Programadores experimentados con esta libertad en la construcción de una solución pueden desarrollar una pieza "hermosa" de código o encontrar una solución "elegante". Sin embargo, para quienes son principiantes y para los

profesores la variedad de soluciones "correctas" puede ser un reto (Fangohr & O’Brien, 2015).

Durante las últimas dos décadas, el interés por convertirse en profesionistas en el área de la Ingeniaría de Software va en aumento. Price y Petre (1997) consideraron que era importante la retroalimentación por parte de los profesores a los estudiantes que aprenden programación, especialmente en tareas electrónicas y las aportaciones al proceso de enseñanza-aprendizaje en línea.

Las herramientas de evaluación automática de las competencias programación, buscan aprovechar las ventajas que el internet proporciona para no realizar instalaciones de entornos de ejecución, es por ello que las herramientas de evaluación automática de competencias de programación se construyen para trabajar en línea.

Los sistemas de evaluación en línea deben de contemplar una serie de características para cada forma de evaluación que implementen. Nikolova (2012) propone cinco funciones principales que deben de contener los sistemas de evaluación en línea: la evaluación, el diagnostico, la formación, la motivación y la trazabilidad, los cuales se pueden ver en la Tabla 2.2.

Tabla 2.2

Funciones principales de los sistemas de evaluación en línea.

Función	Descripción
a) Evaluación (Retroalimentación)	Para medir los resultados procesos del proceso de aprendizaje, también para que el docente y el estudiante puedan hacer una reflexión sobre lo aprendido y las áreas de oportunidad.
b) Diagnostico	Para detectar problemas educativos individuales, ya que el proceso de enseñanza-aprendizaje en esta era digital es necesario personalizarlo lo más posible.
c) Formación	Para dirigir y gestionar el proceso de aprendizaje.
d) Motivación	Estimular y motivar, ya que uno de los grandes problemas de la evaluación es que el estudiante los percibe como algo que tiene que cumplir y no como un elemento importante en su formación que le ayudara a reflexionar sobre su forma de aprender y los conocimientos con los que cuenta.
e) Trazabilidad	Es necesario que los sistemas de evaluación recopilen la mayor cantidad de estadísticas que apoyen a entender como se ha realizado el proceso de aprendizaje y solución de la evaluación por parte del alumno, pues no basta con que el alumno de la respuesta correcta, el camino para llegar a ella siempre aportara información importante para el estudiante y el docente.

Fuente: Elaboración propia con base en Nikolova (2012)

2.6 Métricas de Software

Las métricas de software se han convertido en un punto esencial en la Ingeniería de Software, ya que miden ciertas características del software con el propósito de obtener información para comprobar si los requerimientos son consistentes y completos, o si el diseño es de calidad o simplemente si el código está listo para ser probado (Dávila, 2002).

Los términos *medida, medición y métrica* son diferentes, dentro de la Ingeniería de Software, una medida proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto. La medición es el acto de determinar una medida. Se define formalmente medición como “el proceso por el cual se asignan números o símbolos a los atributos de un proceso o producto, de tal manera que los describan de acuerdo a reglas claramente definidas” (Norman & Lawrence, 1997). Un atributo es una característica o propiedad de un proceso o producto.

El IEEE (1990) en su documento *Standard Glossary of Software Engineering Terms* define métrica como una medida cuantitativa del grado en que un Sistema, componente o proceso que posee un atributo dado. Una métrica por sí sola no representa nada. Es un número que debe ser comparado con alguna norma o estándar para que tenga un significado útil.

Las métricas de calidad del software tradicionales se han centrado fundamentalmente en tres aspectos de evaluación: las métricas de procesos, las métricas de productos y las métricas de recursos (Briand et al., 1996) las cuales se pueden ver en la Tabla 2.3.

Tabla 2.3

Clasificación tradicional de las métricas de software

Métrica	Descripción
a) Métricas de procesos	Cuantifican la conducta del proceso; una categoría principal de ellas es el conteo de eventos, por ejemplo, se cuentan los elementos que ocurren, tal como los números de defectos encontrados en la prueba o los cambios de requerimientos.
b) Métricas de productos	Se refieren al volumen generado. Incluyen líneas de código, páginas de un documento, numero de pantallas, numero de archivos, entre otros. Estas métricas pueden ser de varios elementos del producto, por ejemplo, la cantidad de código que se realizó en la fase de implementación o las líneas de código que fueron modificadas durante la prueba de unidad.
c) Métricas de recursos	Se refieren a las horas laborales. Esto implica a las horas de trabajo, categorías de trabajo y actividades de las tareas. Mientras las métricas comunes de recursos se programan en meses o semanas, las métricas de tiempo personales son muy útiles si están en minutos.

Fuente: Elaboración propia con base en IEEE (1990)

Las métricas buscan primordialmente entender qué ocurre durante el desarrollo de software, controlar qué es lo que ocurre en la aplicación resultante y mejorar los procesos del estudiante y sus aplicaciones.

3. PROBLEMÁTICA

La formación en la Programación Orientada a Objetos no solo es codificar, es la integración de grupos de habilidades para el desarrollo de software. Desarrollar software es un proceso complejo que se ha definido a lo largo de los años, la cual involucra distintas actividades que requieren de las siguientes habilidades y competencias específicas (McConnell, 2004). La evaluación efectiva en el proceso de formación en la Programación Orientada a Objetos no se cumple mediante la técnica de evaluación de caja negra ya que no se permite una correcta valoración de las propiedades del paradigma. Un método de caja blanca permitiría tener una mejor visión de la solución y de que conceptos el alumno está utilizando para ello.

El método de caja blanca implica una gran inversión de tiempo de los docentes, y estas revisiones por el número de alumnos podrían verse afectadas en la calidad debido a la complejidad de analizar diferentes códigos fuentes, por ello se plantea que este método se aplique de manera automática en base al código fuente de cada uno de los estudiantes. Utilizar un método de caja blanca automatizado del código fuente de cada uno de los estudiantes nos permitiría mejorar la asertividad de la evaluación del aprendizaje, dar seguimiento a las características individuales de cada estudiante, proporcionar una retroalimentación personalizada y automática, impulsar el aprendizaje de los estudiantes, apoyar en la impartición del conocimiento a los profesores y permitir a las Instituciones Educativas llevar un control de perfiles.

Por lo tanto, el objetivo general de esta investigación es diseñar un método que permita evaluar, de manera efectiva, el grado de cumplimiento de las

habilidades y competencias de la programación del Paradigma Orientado a Objetos basado en el código fuente, y la hipótesis de investigación “Los componentes de software son capaces de evaluar de manera automática el aprendizaje de la programación orientada a objetos.”

4. PROPUESTA

Partiendo de la metodología de investigación basada en diseño, el entorno de estudio de la problemática es la Facultad de Informática de la Universidad Autónoma de Querétaro. La Facultad de Informática brindó el entorno adecuado ya que es el encargado de la formación académica de los futuros profesionistas en el área de las Tecnologías de la Información y dentro de sus diferentes planes de estudios todos congenian en la impartición de la programación del Paradigma Orientado a Objetos.

4.1 Habilidades y competencias

La identificación de las habilidades y competencias que los alumnos en formación dentro de esta asignatura, se realizó desde dos puntos de vista. La primera es haciendo un análisis de lo que algunas instituciones a nivel internacional proponen en la formación en el área del desarrollo de software y lo que la facultad propone en comparación con estas instituciones. Y la segunda ya es de manera a profundidad con los contenidos dentro de las asignaturas en donde se pueda trabajar.

El *Institute of Electrical and Electronics Engineers* (IEEE) propone un modelo de competencias en la Ingeniería de Software (SWECOM, 2014) en donde se menciona que las personas competentes tienen las habilidades necesarias, organizadas en distintos niveles, para llevar a cabo las actividades de trabajo. En este modelo de competencia el conocimiento es lo que se sabe y las habilidades es lo que se puede lograr hacer. Con esta diferenciación se especifican áreas, habilidades y actividades de trabajo para cada habilidad, las cuales son habilidades

cognitivas, habilidades de destreza y comportamiento y habilidades técnicas, las cuales determinan los conocimientos requeridos y las disciplinas relacionadas, las cuales se pueden encontrar en la Figura 4.1.

Figura 4.1 Elementos del SWECOM. Fuente: Elaboración propia con base en SWECOM (2014)

Cada una de las habilidades del modelo se divide en cinco niveles de competencia: Técnico, Practicante de primer nivel, Practicante, Líder técnico e Ingeniero de programación superior.

Otra institución internacional es *Association for Computing Machinery (ACM) - CCECC (Committee for Computing Education in Community Colleges)*, la cual fue integrada de manera oficial en 1991 como el comité permanente de la junta de educación de ACM al cual llamaron CCECC, esta tiene la misión de apoyar a la

comunidad universitaria y de estudios técnicos en aspectos acerca de la educación en el área de la computación en todo el mundo. La Figura 4.1 muestra como ACM clasifica el área de la informática en cinco sub-disciplinas: Ciencias de la computación, Ingeniería en Computación, Ingeniería de Software, Sistemas de Información y Tecnologías de Información.

Tabla 4.1

Habilidades y competencias de las sub-disciplinas ACM

Disciplina	Habilidades y competencias
Ingeniería Computación	• Diseño y la construcción de sistemas basados en procesadores que comprenden componentes de hardware, software y comunicaciones. • Plan de estudios centrado en la síntesis de la ingeniería eléctrica y la informática aplicada al diseño de sistemas. • Los egresados serán capaces de diseñar e implementar sistemas que involucren la integración de dispositivos de software y hardware.
Sistemas de Información	• Aplicación de principios informáticos a los procesos de negocio, superando los campos técnicos y de gestión. • Plan de estudios centrado en el diseño, implementación y pruebas de sistemas de información aplicados a procesos de negocios (nóminas, recursos humanos, BD corporativas, data warehousing y minería, comercio electrónico, finanzas, la toma de decisiones) • Los egresados serán capaces de analizar requisitos de información y procesos de negocios para especificar y diseñar sistemas que estén alineados con las metas de la organización.

Tecnología de Información	• Diseño, implementación y mantenimiento de soluciones tecnológicas para el apoyo a los usuarios de sistemas. • Plan de estudios centrado en la elaboración de soluciones de hardware y software aplicadas a redes, seguridad, cliente-servidor y computación móvil, aplicaciones web, recursos multimedia, sistemas de comunicaciones, así como la planificación y gestión del ciclo de vida de la tecnología. • Los egresados deben ser capaces de trabajar eficazmente en la planificación, implementación, configuración y mantenimiento de la infraestructura informática de una organización.
Ciencias de la Computación	• Base en fundamentos teóricos de la informática, algoritmos y técnicas de programación, como se aplica a los sistemas operativos, inteligencia artificial, informática y similar. • Los egresados deben estar preparados para trabajar en una amplia gama de posiciones que incluyen tareas desde el trabajo teórico hasta el desarrollo de software.
Ingeniería de Software	• Este plan de estudios se centra en la integración de los principios de la informática con las prácticas de ingeniería aplicadas a la construcción de sistemas de software para aviación, aplicaciones sanitarias, criptografía, control de tráfico, sistemas meteorológicos y similares. Al graduarse, los estudiantes iniciando carreras como ingenieros de software deben ser capaces de realizar adecuadamente y gestionar las actividades en cada etapa del ciclo de vida de los sistemas de software a gran escala.

Fuente: Elaboración propia con base en CCECC (2017)

Se realizó un análisis de las cinco sub-disciplinas que propone ACM y los perfiles que la Facultad de Informática ofrece. Se muestra en la Tabla 4.2, una relación de concordancia de habilidades y competencias entre las sub-disciplinas y los planes de estudio.

Tabla 4.2

Relación de sub-disciplinas de ACM y planes de estudio de la Facultad de Informática

	Ingeniería Computación	Sistemas de Información	Tecnología de Información	Ciencias de la Computación	Ingeniería software
Ing. software UAQ				✓	✓
Lic. Informática UAQ		✓			
Ing. Computación UAQ	✓			✓	
Ing. telecomunicaciones UAQ					
Lic. Admón. de las tecnologías de información UAQ			✓		

Fuente: Elaboración propia.

Finalmente, dentro de la Facultad, dentro de cada plan de estudios se analizó las concordancias con las materias en las cuales la evaluación del código fuente puede ser aplicada, teniendo los resultados de la Tabla 4.3

Tabla 4.3

Relación de materias de los planes de estudios de la Facultad de Informática relacionados a la evaluación de código

	Ing. software UAQ	**Lic. Informática UAQ**	**Ing. Computación UAQ**	**Lic. Admón. de las tecnologías de información UAQ**	**Ing. Telecomunicaciones UAQ**
Introducción a la Programación	✓	✓	✓	✓	✓
Programación Orientada a Objetos	✓	✓	✓	✓	✓
Algoritmos y estructuras de datos	✓	✓	✓		✓
Desarrollo de Aplicaciones con Acceso a datos	✓	✓		✓	
Paradigmas de programación	✓			✓	
Desarrollo web	✓	✓		✓	
Programación de dispositivos móviles	✓				
Arquitectura y desarrollo de sist. Distribuidos	✓				

Fuente: Elaboración propia

4.2 Competencias en la programación en el Paradigma Orientado a Objetos

El desarrollo de software es una tarea que fundamentalmente se hace de forma manual, sin embargo, las aplicaciones de software son cada vez más complejas, es por ello que buscando una metodología de desarrollo que pudiera construir aplicaciones modelando la realidad se creó la metodología de programación orientada a objetos. Las aplicaciones construidas bajo el paradigma orientado a objetos, tienen las características adecuadas para expresar la complejidad de un sistema, las cuales son: adaptabilidad, reusabilidad y mantenibilidad (Durán, Gutiérrez, & Pimentel, 2007).

Para poder construir aplicaciones adaptables, reusables y mantenibles, los lenguajes de programación orientados a objetos deben soportar una serie de características, tales como la abstracción para definir y extraer las propiedades del objeto, encapsulación que permite solo el acceso a través de la interfaz de usuario, modularidad para dividir un objeto en piezas más pequeñas, el principio de ocultación ya que cada módulo está aislado del exterior, el polimorfismo, comportamiento diferente asociado a distintos objetos con el mismo nombre y la herencia ya que las clases no están aisladas si no que se relacionan entre si formando clasificaciones jerárquicas.

Los lenguajes de programación orientados a objetos, manejan una serie de conceptos clave a través de los cuales pueden soportar las características que los definen dentro del paradigma orientada a objetos los conceptos clave son: Clase que es una plantilla desde la que se pueden crear objetos, Objeto es una instancia de una clase, Miembros de datos o variables de clase, Método o función construida

en una clase y Herencia que establece relaciones jerárquicas de especialización entre clases (Holzner, 2000).

La programación fomenta la construcción de conocimientos mediante apropiación, ya que los resultados son considerados como propios, además, los objetos y los artefactos construidos materializan el pensamiento y, simultáneamente, lo modifican y enriquecen. Es por esta razón que la programación desarrolla las siguientes competencias: Capacidad para expresarse con tecnologías; Habilidad para colaborar, comunicar y trabajar en equipos; Capacidad para resolver problemas complejos; Habilidad para desarrollar, planificar y ejecutar proyectos complejos; Desarrollar la autoestima y la autoconfianza y Creatividad (Berrocoso, Sánchez, & Arroyo, 2015).

Formar para la construcción de software en un paradigma orientado a objetos es más que solo codificar, desarrollar software es un proceso complejo que se ha definido a lo largo de los años, el cual involucra distintas actividades que requieren de grupos de habilidades y competencias específicas (McConnell, 2004). Estas actividades son: Análisis, Diseño, Implementación, Ejecución y Evaluación. Las competencias derivadas de estas actividades son: Análisis, leer y entender especificaciones, identificar deficiencias en la especificación, generar una estrategia en base a la especificación; Diseño, Desarrollar un conjunto de clases funcional, identificar clases redundantes, Identificar los métodos y funciones de cada clase. Identificar las relaciones entre clases; Implementación, Desarrollar pruebas, crear conjuntos de datos de prueba; Ejecución, Ejecutar pruebas, documentar resultados; Evaluación Identificar errores en el programa a partir de los resultados, depuración del programa (Jones, 2001).

4.3 Métricas propuestas

Las métricas de calidad de software fueron creadas para cuantificar los productos y fueron diseñadas para medir aspectos dentro de la industria del software. Esta propuesta busca crear un método de evaluar las propiedades del paradigma orientado a objetos en un entorno académico de evaluación. Las métricas de software propuestas se dividen en tres grupos principales, métricas de compilación, métricas de codificación y métricas de ejecución como se ve en la Figura 4.2.

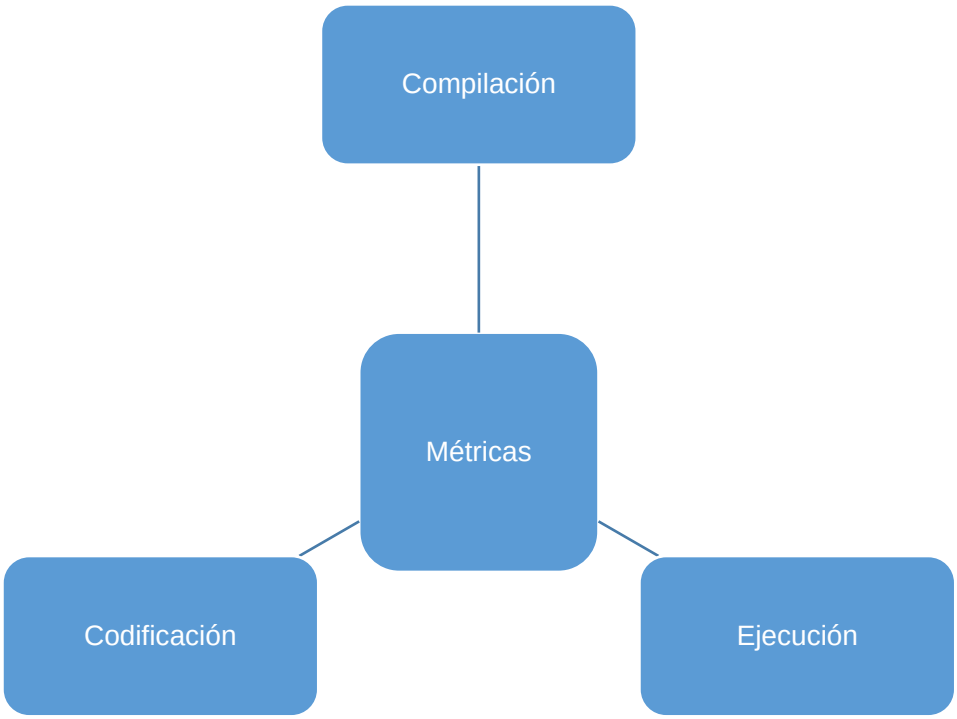

Figura 4.2 Clasificación propuesta de las métricas de software

En la Tabla 4.4 se muestra las características principales de cada tipo de métricas según la prepuesta planteada.

Tabla 4.4

Clasificación del grupo de métricas propuesto

Métrica	Descripción
a) Métricas de codificación	Se refiere a las características que obtenemos al escribir el código y darle solución a un problema. Las métricas dentro de esta categoría son: • Número de operadores distintos • Número de operandos distintos • Total de operadores • Total de operandos • Vocabulario del programa • Longitud del programa • Volumen • Volumen Potencial • Nivel de programa • Dificultad del programa • Inteligencia contenida en el programa • Esfuerzo • Tiempo total de producción • Tiempo total de prueba • Tiempo total de corrección • Tiempo total del producto

b)	Métricas de ejecución	Se refiere a las características que obtenemos al ejecutar una solución parcial. Las métricas dentro de esta categoría son: • LOC base • Agregado • Modificado • Suprimido • Reutilizado • Nueva reutilización • LOC nuevo y cambiante • LOC total • Número total de defectos • Número total de defectos Corregidos • Número de defectos encontrados y no corregidos • Número de defectos por tamaño del producto • Número de defectos encontrados por hora en base al tiempo toral del producto • Número de defectos corregidos por hora en base al tiempo total del producto • Número de defectos encontrados por hora en base al tiempo total de la actividad de pruebas. • Número de defectos corregidos por hora en base al tiempo total de la actividad de correcciones.
c)	Métricas de compilación	Se refiere a las características que obtenemos al compilar la solución final. Las métricas dentro de esta categoría son: • Densidad de defectos • Proyección de futuros defectos • Número Ciclomático

Fuente: Elaboración propia

4.3.1 Métricas de Codificación

a) Número de operadores distintos

Cantidad de símbolos distintos que afectan el valor u orden del operando.

$n1 = número\ de\ operadores\ distintos$

b) Número de operandos distintos

Cantidad de variables o constantes distintas que se utilizan durante la implementación.

$n2 = número\ de\ operandos\ distintos$

c) Total de operadores

Cantidad total de símbolos que afectan el valor u orden del operando.

$N1 = Total\ de\ operadores$

d) Total de operandos

Total, de variables o constantes que se utilizan durante la implementación.

$N2 = Total\ de\ operandos$

e) Vocabulario del programa

Número de los diferentes operandos y operadores usados para construir un programa.

$n = n1 + n2$

Donde:

n: Vocabulario del sistema

n1: número de operadores distintos

n2: número de operandos distintos

f) Longitud del programa

La longitud en términos del número total de operadores y operandos utilizados durante la implementación.

$N = N1 + N2$

Donde:

N: Longitud del programa (Total de operadores y operandos)

N1: Total de operadores

N2: Total de operandos

Indicadores:

N <= 100 pequeños

N > 100 y N <= 500 medianos

N > 500 y N <= 1500 grandes

N > 1500 muy grandes

La longitud en términos de número de operadores y operandos diferentes.

$\widehat{N} = n1\ log_2\ n1 + n2\ log_2\ n2$

Donde:

$\widehat{N}$: Longitud del programa (operadores y operandos distintos)

n1: número de operadores distintos

n2: número de operandos distintos

Indicadores:

$\widehat{N}$ > 100 y $\widehat{N}$ < 2000 pequeños

$\widehat{N}$ > 4000 grandes

g) Volumen

Calcula el volumen de un algoritmo. El volumen de los algoritmos depende según el lenguaje. La interpretación de un volumen de programa con dimensión en bits, buscando obtener el número de bits requeridos para especificar un programa.

$V = N\ log_2 n$

Donde:

V: Volumen potencial

N: Longitud del programa

n: Vocabulario del programa

h) Volumen potencial

Volumen mínimo potencial para un programa.

$V* = (n1* + n2*) log_2 (n1* + n2*)$

Donde:

V*: Volumen potencial

n1*: Número mínimo posible de operadores

n2*: Número de los diferentes parámetros de entrada/salida.

El número mínimo posible de operadores n1* para cualquier algoritmo es conocido. Este debe consistir en un operador diferente para el nombre de la función o procedimiento y otro para servir como una asignación o grupo de símbolos. Por tanto n1*=2.

$V* = (2 + n2) log_2 (2 + n2*)$

Donde:

V*: Volumen potencial

n2*: Número de los diferentes parámetros de entrada/salida.

i) Nivel del programa

Da una idea del nivel de detalle con que ha sido codificado. Se entiende que cuanto más código se una para una función dada, de más bajo nivel será. En el límite, la llamada a función tiene el nivel más alto, ya que su volumen real coincide con el potencial.

$L = V*/V$

Donde:

L: Nivel de programa

V*: Volumen potencial

V: Volumen del programa

Ya que el volumen potencial frecuentemente no está disponible se tiene la siguiente formula alterna:

$L = (\frac{2}{n1})(\frac{n2}{N2})$

Donde:

L: Nivel del programa

n1: Número de operadores distintos

n2: Número de operandos distintos

N2: Total de operandos

El nivel de programa depende del lenguaje que está utilizando, variando así también para programas equivalentes en el mismo lenguaje ya que depende de la experiencia y estilo del programador.

Indicadores:

Su valor varía entre 0 y 1. Si L=1 el programa está escrito en el más alto nivel posible.

j) Dificultad del programa

Lo inverso al nivel del programa. Cuando el volumen crece, el nivel del programa decrece y la dificultad se incrementa. Las prácticas de programación como uso redundante de operandos o error de usar frases de control de nivel más alto tenderán a incrementar el volumen, así como la dificultad.

$D = 1/L$

Donde,

D: Dificultad del programa

L: Nivel de programa

Indicadores:

Si el nivel de programa decrece la dificultad se incrementa. D= 1 significa que el programa está escrito en el más alto nivel y su dificultad es nula.

k) Inteligencia contenida en el programa

Con el volumen y el nivel del programa, se calcula la inteligencia del programa. Este valor se correlaciona con el tiempo total de programación y depuración. Esta métrica permite estimar la complejidad desde el punto de vista del tiempo de programación y la depuración. Esta métrica permanece invariante a cambios en el lenguaje de programación, por lo que permite medir las ventajas y desventajas del código empleado.

$I = \hat{L}V$

Donde:

I: Inteligencia contenida del programa

L: Nivel de programa

V: Volumen del programa

l) Esfuerzo

El esfuerzo es necesario para producir una posición de software está relacionado con la dificultad de entenderlo. Por lo tanto, el esfuerzo puede ser usado como una medida de la claridad del programa.

$E = V/L$

Donde:

E: Esfuerzo

V: Volumen del programa

L: Nivel de programa

m) Tiempo total de producción

Tiempo empleado durante la codificación

$TDC = TDC_{final} - TDC_{inicial}$

Donde:

TDC: Tiempo de codificación

TDC_{final}: Punto de termino

TDC_{inicio}: Punto de inicio

n) Tiempo total de prueba

Tiempo total empleado para las pruebas del código.

$TP = TP_{final} - TP_{inicial}$

Donde:

TP: Tiempo de pruebas

TP_{final}: Punto de termino

TP_{inicio}: Punto de inicio

o) Tiempo total de corrección

Tiempo total empleado para la corrección de los defectos encontrados durante las pruebas.

$TC = TC_{final} - TC_{inicial}$

Donde:

TC: Tiempo de correcciones

TC_{final}: Punto de termino

TC_{inicio}: Punto de inicio

p) Tiempo total del producto

Tiempo total donde se considera la codificación, las pruebas y la corrección.

$TT = TDC + TP + TC$

Donde:

TT: Tiempo total del proyecto

TC: Tiempo de correcciones

TDC: Tiempo de codificación

TP: Tiempo de pruebas

4.3.2 Métricas de Ejecución

a) LOC Base

Tamaño de la versión original sin modificación

$LOC_{base} = número\ de\ líneas\ de\ código\ de\ la\ versión\ original$

b) LOC Agregado

Líneas de código para un nuevo programa o agregado a uno existente.

$LOC_{agregado}$

$= número\ de\ líneas\ de\ código\ de\ un\ nuevo\ programa\ o\ agregado$

c) LOC Modificado

Líneas de código de un programa existente que sufre modificaciones

$LOC_{modificado} = número\ de\ líneas\ de\ código\ modificadas$

d) LOC Suprimido

Líneas de código para un programa existente que se suprime.

$LOC_{suprimido} = número\ de\ líneas\ de\ código\ que\ se\ suprimen$

e) LOC Reutilizado

Líneas de código que se toman de una librería, sin hacer modificaciones a un programa nuevo o una versión previa.

$LOC_{reutilizado} = número\ de\ líneas\ de\ código\ reutilizadas$

f) LOC nueva reutilización

Líneas de código que se desarrollan y contribuyen a una librería de reutilización

$LOC_{nuReutilizacion} = número\ de\ líneas\ de\ código\ nuevas\ en\ una\ libreria$

g) LOC nuevo y cambiante

$LOC_{nuevoCambiante} = LOC_{agregado} + LOC_{modificado}$

h) Número total de defectos

Defectos totales encontrados durante la ejecución.

$D = Número\ total\ de\ defectos$

i) Número total de defectos corregidor

Defectos totales corregidos.

$DC = Número\ total\ de\ defectos\ corregidos$

j) Número de defectos encontrados y no corregidos

Defectos totales encontrados, pero no corregidos.

$DENC = D - DC$

Donde:

DENC: Defectos encontrados y no corregidos

D: Total de defectos

DC: Total de defectos corregidos

k) Número de defectos por tamaño del producto

Total, de defectos por tamaño del producto

$DT = D - LOC$

Donde:

DT: Numero de defectos por tamaño del producto

D: Total de defectos

LOC: Total de líneas de código

l) Número de defectos encontrados por hora en base al tiempo total del producto

Total de defectos por hora en base al tiempo total del producto.

$DEH = D/TT * 60$

Donde:

DEH: Defectos por hora en base al tiempo del producto

D: Total de defectos

TT: Tiempo total del proyecto

m) Número de defectos encontrados por hora en base al tiempo total de la actividad de pruebas.

Total de defectos encontrados por hora en base al tiempo de la actividad de pruebas

$DEH_{prueba} = D/TP * 60$

Donde:

DEH_{prueba}: Defectos encontrados por hora en base al tiempo de la actividad de pruebas

D: Total de defectos

TP: Tiempo de pruebas

n) Número de defectos corregidos por hora en base al tiempo total de la actividad de correcciones.

Total de defectos corregidos por hora en base al tiempo de la actividad de correcciones

$$DEH_{correcciones} = DC/TC * 60$$

Donde:

$DEH_{correcciones}$: Defectos corregidos por hora en base al tiempo de la actividad de correcciones.

DC: Total de defectos corregidos

TC: Tiempo de correcciones

4.3.2 Métricas de Compilación

a) Densidad de defectos

La densidad de defectos se refiere a los defectos por KLOC nuevo y cambiante encontrado en un programa.

$Dd = 1000 * D/LOC$

Donde:

Dd: Densidad de defectos

D: Total de defectos

LOC: Total de líneas de código

La densidad del defecto se mide para el proceso entero de desarrollo y para algunas fases del proceso específicas. La fase de pruebas se limita a quitar solamente una fraccione de los defectos en un producto, cuando en realidad hay más defectos que se incorporan a la fase de pruebas pero en su versión final. Por lo tanto, el número de defectos que se quedan en el producto después de que las pruebas terminan.

Si un programa de 150 LOC tuviera 18 defectos, la densidad del defecto seria: 1000 * 18 / 150 = 120 defectos/KLOC

Indicadores:

En PSP, un programa con cinco o menos defectos/KLOC es considerado como un programa de buena calidad.

b) Proyección de futuros defectos

Número de líneas de código que se proyecta tendrá el programa

$$Dd_{plan} = 100(D_1 + \cdots + D_i)/(N_1 + \cdots + N_i)$$

Donde:

Dd_{plan}: Densidad de defectos planeados

D: Total de defectos

N: Número de líneas de código

i: número de programas anteriores

Porcentaje de defectos/KLOC de los programas previamente desarrollados al proyectado. Con la proyección de futuros defectos y la estimación de las próximas líneas de código se calcula el numero esperado de defectos.

$$D_{plan} = N_{plan} * Dd_{plan}$$

Donde:

Dd_{plan}: Densidad de defectos planeados

N_{plan}: Número de líneas de código estimadas

D_{plan}: Número de defectos encontrados

c) Número ciclomático

Número mínimo de caminos necesario para, mediante combinaciones, construir cualquier otro camino presente en el grafo. Codificado en cualquier lenguaje, puede demostrarse su número ciclomático.

$V(G) = a - n + 2c$

Donde:

a: Número de arcos

n: Número de nodos

c: Número de componentes conectados (normalmente c=1)

$V(G) = N + 1$

Donde:

N: Número total de sentencias de decisión en el programa

1: Camino natural

5. EVALUACIÓN

Para validar el método de evaluación propuesto se tomó un grupo de 25 alumnos a quienes utilizando el modelo de Proceso Personal de Software se les evaluó su desempeño y cumplimiento de las propiedades para lo que los ejercicios fueron creados haciendo uso de las métricas de software seleccionadas.

La práctica actual de la ingeniería de software implica una disciplina de la ingeniería estructurada. Existen Principios de gestión de calidad conocidos y efectivos que pueden ser aplicados al software, el más conocido es el Proceso Personal de Software (PSP).

El principal objetivo del Proceso Personal de Software es hacer Ingenieros de Software conscientes de los procesos que se utilizan para hacer su trabajo, y el desempeño de esos procesos. Los profesionales del software establecen metas personales, definen los métodos a utilizar, miden su trabajo, analizan los resultados, y ajustan sus métodos para cumplir sus objetivos. Es una estrategia para el autodesarrollo profesional y mejorar la eficacia (Humphrey, 1995).

El PSP se basa en los siguientes principios: la calidad del sistema de software se rige por la calidad de los peores componentes, la calidad de un componente de software es dirigido por la persona que lo desarrolló en base a su conocimiento, disciplina y compromiso, y finalmente el software. Los practicantes deben conocer su propio desempeño y medida así como rastrear y analizar el trabajo para aprender de las variaciones de rendimiento e incorporar estas lecciones en las prácticas personales.

Lograr un Proceso Personal de Software estable permite al ingeniero de software estimar y planificar el trabajo, cumplir los compromisos, identificar las habilidades, e identificar áreas en necesidad de mejora. Un buen PSP también proporciona a los profesionales de software una base para desarrollar y practicar la fuerza industrial. Este proceso marca la pauta a fomentar la disciplina personal, la cual permite mostrar cómo mejorar el desempeño personal, y mejorar continuamente la productividad, calidad y previsión en el trabajo.

Para el análisis de los resultados de las evaluaciones se dividieron en las siguientes variables de evaluación:

- Tamaño del programa
- Tiempo de desarrollo
- Número de errores
- Tiempo estimado de corrección de errores
- Número de veces de compilación

A continuación mostramos la gráficas con los datos estadísticos de cada una de esta variables.

La Figura 5.1 muestra el tamaño promedio por programa, tomando en consideración las mediciones de los alumnos en el grupo.

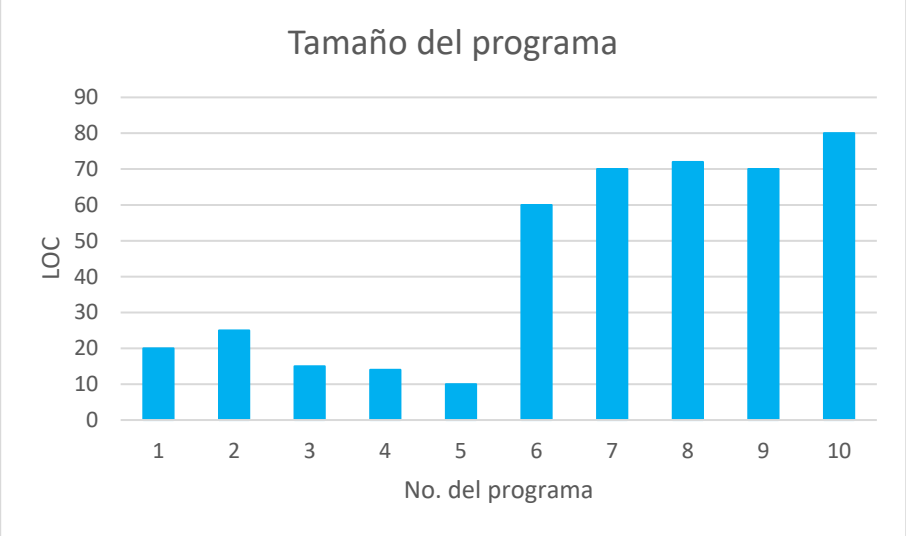

Figura 5.0.1 Tamaño promedio por programa. Fuente: Elaboración propia

La Figura 5.2 muestra el tiempo de desarrollo por programa, tomando en consideración las mediciones de los alumnos en el grupo.

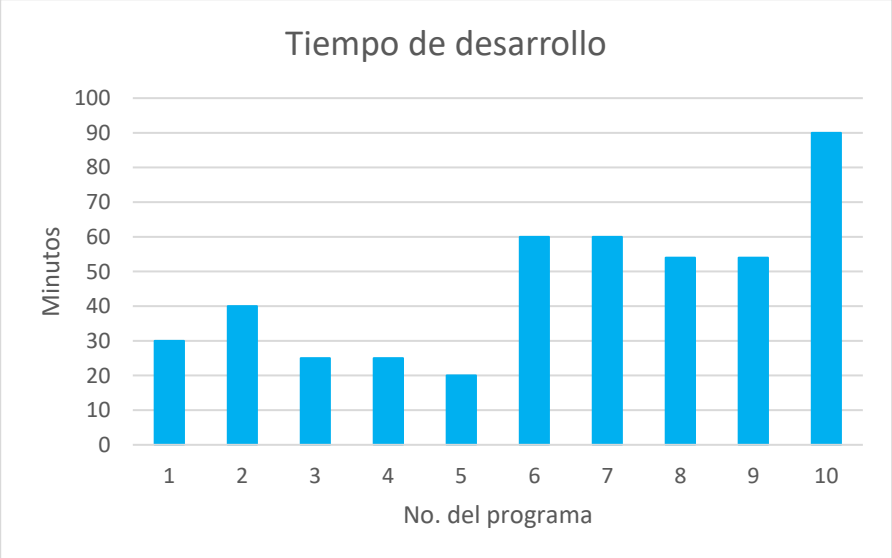

Figura 5.0.2 Tiempo de desarrollo promedio por programa. Fuente: Elaboración propia

La Figura 5.3 muestra el número estimado de errores por programa, tomando en consideración las mediciones de los alumnos en el grupo.

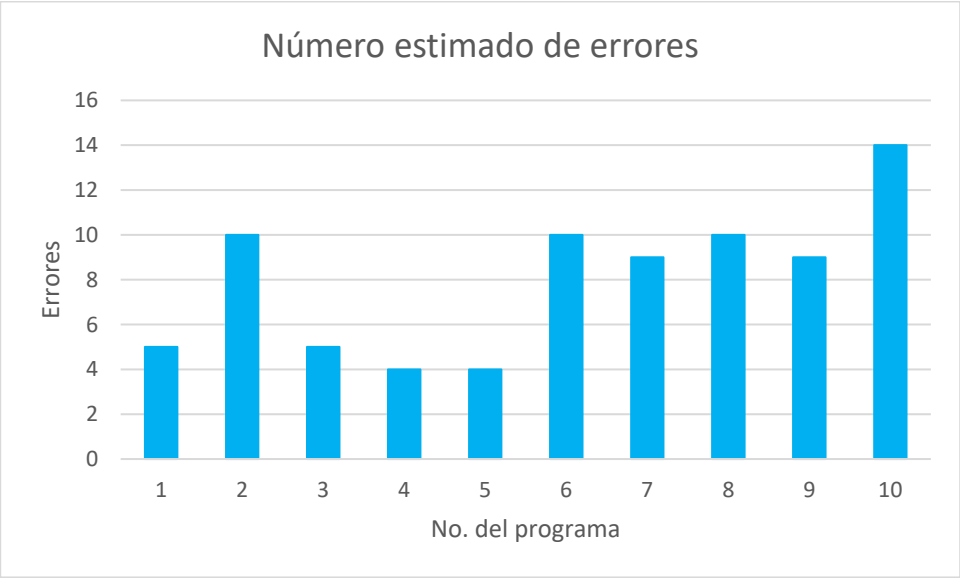

Figura 5.0.3 Número estimado de errores promedio por programa. Fuente: Elaboración propia

La Figura 5.4 muestra el tiempo estimado en corrección de errores por programa, tomando en consideración las mediciones de los alumnos en el grupo.

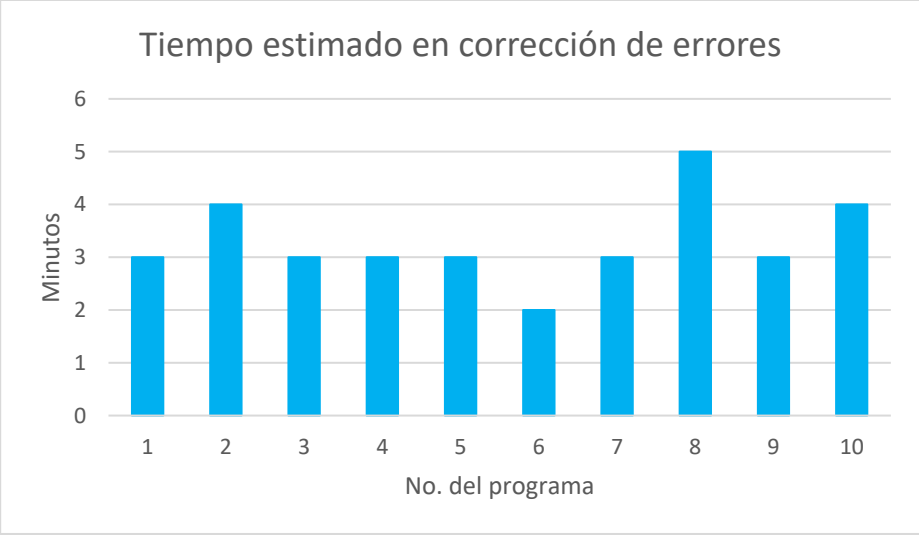

Figura 5.0.4 Tiempo estimado en corrección de errores promedio por programa. Fuente: Elaboración propia.

La Figura 5.5 número de compilaciones por programa, tomando en consideración las mediciones de los alumnos en el grupo.

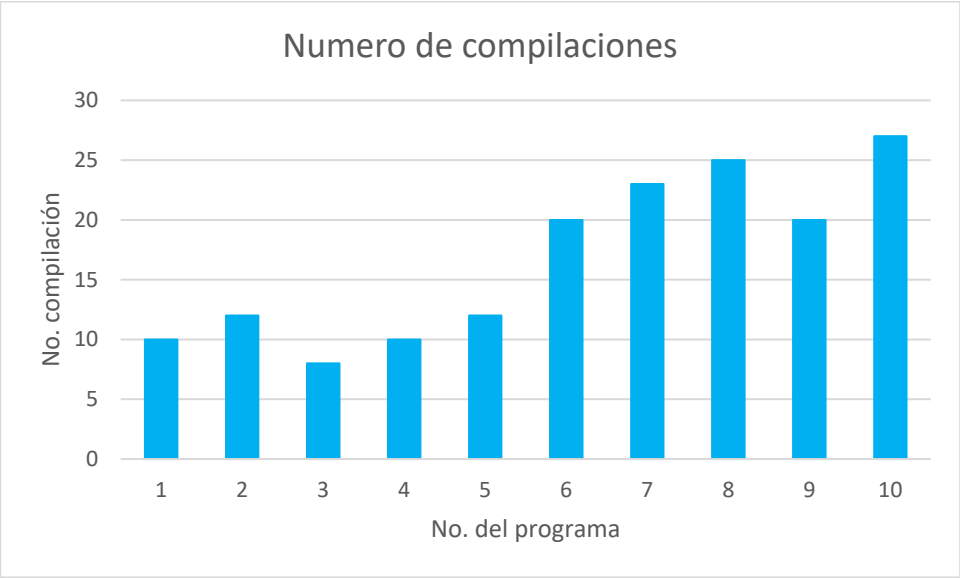

Figura 5.5 Tiempo estimado en corrección de errores promedio por programa. Fuente: Elaboración propia.

6. CONCLUSIONES Y TRABAJOS FUTUROS

Este apartado resume las conclusiones que se pueden obtener una vez concluido el trabajo de investigación realizado. Al finalizar el proyecto, podemos hacer un balance de los objetivos cumplidos a partir de los objetivos propuestos al inicio del proyecto; así como los nuevos objetivos que han ido surgiendo en las distintas etapas por las que ha pasado el proyecto para solventar los distintos problemas encontrados en el desarrollo.

Por otro lado, tenemos que tener en cuenta que este proyecto, puede ofrecer mucho más, es decir, existen numerosos caminos por los que podemos seguir investigando para conseguir una herramienta de evaluación mucho más robusta y capaz de ofrecer detalles más puntuales.

Como objetivos iniciales del proyecto se propusieron los siguientes: Construir un modelo de habilidades y competencias requeridas en la formación del Paradigma Orientado a Objetos. Elaborar un mapa de asociaciones entre propiedades del Paradigma Orientado a Objetos con las habilidades y competencias en la formación de este mismo paradigma; que permita determinar el aprendizaje de los estudiantes. Desarrollar un modelo de métricas para evaluar la presencia o ausencia de las propiedades basado en un código fuente.

El primer y segundo objetivo se consiguieron en la primera fase del proyecto al investigar el estado del arte de los perfiles para el área de programación en la Ingeniería de Software y relacionarlos con las propiedades del paradigma orientado a objetos; mientras que, para conseguir el tercer objetivo, se buscaron las métricas

de software más importantes y apropiadas para evaluar las habilidades y competencias ya establecidas.

A partir del trabajo realizado, se pueden considerar distintos caminos que se pueden seguir para continuar y mejorar el proyecto. La primera es mejorar el modelo e implementar una herramienta con las funcionalidades de evaluación. Esta herramienta sería capaz de añadir nuevos parámetros por métrica que permita al usuario establecer el máximo y mínimo permitido, por ejemplo, máximo y mínimo de métodos por clase. Interpretación de resultados, es decir, una vez obtenidos los resultados de cada una de las métricas ejecutadas, ofrecer al usuario una interpretación de las mismas, indicando al usuario si debe mejorar algún aspecto del código fuente. Contemplar el análisis del código fuente para obtener un código fácilmente interpretable, es decir, permitir al usuario definir reglas que se deben cumplir en el código fuente, por ejemplo, que los nombres de clases empiecen por mayúsculas, que los métodos tengan asociado un comentario inicial.

REFERENCIAS

Alan, R., (2004). Design science in information systems research*. MIS Quarterly.*

Albert, J. (2005). El estudio de los estilos de aprendizaje desde una perspectiva vigostkiana: una aproximación conceptual. *Revista Iberoamericana de Educación.*

Fernández, M. (2005). Propuesta de indicadores del proceso de enseñanza/aprendizaje en la formación profesional en un contexto de gestión de calidad total. *Revista Electrónica de Investigación y Evaluación Educativa.*

Booch, G. (1996). Análisis y diseño orientado a objetos con aplicaciones. *Addison Wesley.*

Santos A., Parra M. (2009). Lineamientos de evaluación del aprendizaje.

Codeval. Most Popular Coding Languages of 2016. (2016). Codeval. Obtenido el 17 de Agosto de 2017 desde http://blog.codeeval.com/

Janke, S. W., Philipp B. (2015). Does outside-in teaching improve the learning of object-oriented programming? *EEE/ACM 37th IEEE International Conference on Software Engineering.*

Kaila, E. L., (2016). Redesigning an object-oriented programming course. *ACM Transactions on Computing Education.*

IEEE. (2014). Software engineering competency model. *IEEE.*

Corral y Balcells. (2014). A game-based approach to the teaching of object-oriented programming languages. *ELSEVIER.*

Malinowski Gadja, E. (2011). Enseñanza de cursos de la programación orientada a objetos para los principiantes. *Revista Ingeniería.*

Aguilar, A., ArielL Rodolfo Aalcantara-Eguren. (2009). La medición del aprendizaje del alumno, a través de la asignación de calificaciones un análisis en la universidad iberoamericana puebla. *X CONGRESO NACIONAL DE INVESTIGACIÓN EDUCATIVA.*

McConnell, S. (2004). Walcome to software construction. En Microsoft (Ed.), (p. 3-8). Microsoft.

Moreno, H. E. R. (2004). La evaluación del aprendizaje: Una propuesta de evaluación basada en productos académicos. *REICE - Revista Electrónica Iberoamericana sobre Calidad, Eficacia y Cambio en Educación.*

Ottogali Alexandra, L. G., Martinez Antonio. (2011). Una notación algorítmica estándar para la programación orientada a objetos. *TELEMATIQUE.*

Rodas, J. M. (2004). Investigación sobre métodos de enseñanza aprendizaje. Facultad de Ingeniería - Universidad Rafael Landívar. Sommerville. (1992). Software engineering. *Addison Wesley.*

Sommerville. (1992). Software engineering. *Addison Wesley.*

Vidya M. Ayer, S. M., y Toby, B. H. (2014). Why scientists should learn to program in python. *CRYSTALLOGRAPHY EDUCATION.*

Yan, L. (2009). Teaching object-oriented programming with games. IEEE Xplore.

A. Robins, J. Rountree, and N. Rountree, “Learning and teaching programming: A review and discussion,” Computer Science Education, vol. 13, no. 2, pp. 137–172, 2003. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1076/csed.13.2.137.14200

Fangohr, h & O’Brien, N. (2015). Teaching Python programming with automatic assessment and feedback provision. Faculty of Engineering and the Environment University of Southampton.

Daly, C., & Waldron, J. (2004). Assessing the Assessment of Programming Ability. SIGCSE Bull., 36(1), 4. https://doi.org/10.1145/971300.971375

Moreno, T. (2016). Evaluación del aprendizaje y para el aprendizaje: reinventar la evaluación en el aula. Ciudad de México: Universidad Autónoma Metropolitana, Unidad Cuajimalpa, División de Ciencias de la Comunicación y Diseño.

Nikolova, M. (2012). Characteristics and Forms of the Electronic Assessment of the Knowledge, 93-98.

De Benito, B. (2006). Diseño y validación de un instrumento de selección de herramientas para entornos virtuales basado en la toma de decisiones multicriterio (Tesis doctoral inédita). Universitat de les Illes Balears,

Palma de Mallorca. Design-Based Research Collective. (2003). Design-Based Research: An Emerging Paradigm for Educational Inquiry. Educational Researcher, 32(1), 5-8.

Humphrey, W. S., A Discipline for Software Engineering, (Reading, MA: Addison-Wesley, 1995).