Mejores prácticas y criterios de calidad en el proceso de desarrollo de código en los cursos de programación en la enseñanza superior

Yedid Erandini Niño Membrillo

Maestra en Ciencias de la Computación y profesora a tiempo completo responsable de la Coordinación de Informática Administrativa del Centro Universitario UAEM Texcoco de la Universidad Autónoma del Estado de México
yeninom@uaemex.mx

María del Rocío Morales Salgado

Doctora en Tecnologías de Información y Análisis de Decisiones y directora del posgrado en Tecnologías de Información y Ciencia de Datos de la Universidad Popular Autónoma del Estado de Puebla
mariadelrocio.morales@upaep.mx

Sodel Vázquez Reyes

Doctor en Ciencias Computacionales y profesor investigador de la Universidad Autónoma de Zacatecas
vazquezs@uaz.edu.mx

Bárbara Emma Sánchez Rinza

Doctora en Física y profesora investigadora de la Facultad de Ciencias de la Computación de la Benemérita Universidad Autónoma de Puebla
brinza@hotmail.com

Extracto

El interés de este estudio radica en contribuir a investigar las dificultades que presentan los estudiantes en los cursos de programación, quienes invierten tiempo al desarrollar programas con deficiencias. Por esta razón, este artículo ofrece una perspectiva de la situación de la programación en la educación superior en México basándose en entrevistas realizadas a docentes que imparten clases de programación en diferentes universidades de la República Mexicana, así como en estudios efectuados en instituciones de educación superior (IES) sobre el tema de la programación. Asimismo, se expone la investigación realizada sobre las prácticas de ingeniería de *software* y los criterios de calidad (CC) utilizados en los cursos de programación en el ámbito universitario, considerando a docentes de universidades públicas. El objetivo de este estudio es establecer un conjunto de buenas prácticas, administrativas y de ingeniería, y CC en el proceso de desarrollo, en el ciclo de vida (CV), para mejorar los cursos de programación y para alentar a los estudiantes a adoptar buenas prácticas en las primeras fases de su formación profesional. Lo anterior contribuirá a que mejoren su proceso de desarrollo y a que creen programas con resultados óptimos.

Palabras clave: prácticas de programación; criterios de calidad (CC); ciclo de vida (CV); desarrollo de programas; educación superior.

Fecha de entrada: 11-01-2020 / Fecha de revisión: 17-02-2020 / Fecha de aceptación: 25-02-2020

Cómo citar: Niño Membrillo, Y. E., Morales Salgado, M.ª R., Vázquez Reyes, S. y Sánchez Rinza, B. E. (2020). Mejores prácticas y criterios de calidad en el proceso de desarrollo de código en los cursos de programación en la enseñanza superior. *Tecnología, Ciencia y Educación*, *17*, 97-126.

Best practices and quality criteria in the coding development process in higher education programming courses

Yedid Erandini Niño Membrillo

María del Rocío Morales Salgado

Sodel Vázquez Reyes

Bárbara Emma Sánchez Rinza

Abstract

The interest of this study lies in contributing to investigate the challenges that programming students face, who invest time developing poor programs. For this reason, this article offers a perspective of the programming situation in higher education in Mexico based on interviews conducted to teachers who teach programming classes in different colleges in Mexico, as well as studies made in higher education institutes (HEI) regarding programming subjects. Likewise, an investigation about software engineering practices and quality criteria (QC) used in the programming courses in the university environment, considering teachers from public universities, is presented. The objective of this study is to stablish a set of good practices, both administrative and engineering, and QC in the development process, in the life cycle (LC), to improve programming courses and encourage the students to adopt good practices in the early stages of their professional training. The above will contribute to improve their development process and will help create better programs with optimal results.

Keywords: programming practices; quality criteria (QC); life cycle (LC); programs development; higher education.

Citation: Niño Membrillo, Y. E., Morales Salgado, M.ª R., Vázquez Reyes, S. and Sánchez Rinza, B. E. (2020). Best practices and quality criteria in the coding development process in higher education programming courses. *Tecnología, Ciencia y Educación*, *17*, 97-126.

Sumario

1. Introducción

Las prácticas de ingeniería de *software* que se realizan en los cursos de programación son relevantes debido a que se centran en el proceso de desarrollo del *software*, el cual se encuentra presente en la industria del *software*, donde estudiantes y egresados de las universidades contribuyen a dar solución a las necesidades informáticas cuando entran a formar parte del mercado laboral. Sin embargo, uno de los desafíos a los que se enfrenta la industria del *software* es que se utilicen buenas prácticas que permitan desarrollar *softwares* que no solo realicen lo que requiere el cliente, sino que den valor a los usuarios.

Las prácticas de ingeniería de *software* que se realizan en los cursos de programación son relevantes debido a que se centran en el proceso de desarrollo del *software*, el cual se encuentra presente en la industria del *software*, donde estudiantes y egresados de las universidades contribuyen a dar solución a las necesidades informáticas cuando entran a formar parte del mercado laboral

En este contexto, las universidades en las que se imparten carreras afines a las ciencias computacionales se han preocupado por incluir en sus planes de estudio materias relacionadas con la programación que, aunque no son determinantes, sí contribuyen a estos resultados. Así, en la programación, no solo se trata de escribir código, sino también se debe analizar el problema que hay que resolver para identificar sus componentes clave, como datos y procesos que serán utilizados por los estudiantes para que con su creatividad den una propuesta de solución al problema (Romero, Lepage y Lille, 2017). Sin embargo, a los estudiantes les resultan difíciles las materias relacionadas con la programación debido a la comprensión deficiente de los conceptos, al escaso entendimiento de los problemas planteados, a la dificultad de identificar las estructuras que hay que emplear, a la escasa comprensión de la semántica y al desconocimiento de la sintaxis del lenguaje de programación que tienen que utilizar, resultando algo evidente que hay poca planificación del proceso en la resolución de problemas (Bosse y Gerosa, 2017; Olier, Gómez y Caro, 2017; Ozmen y Altun, 2017).

Lo anteriormente expuesto no es algo nuevo. De hecho, desde finales de los años sesenta del siglo XX se ha observado empíricamente que los estudiantes de programación aprenden de manera muy limitada porque su formación se centra más en el lenguaje que en el desarrollo de la lógica, a lo que se da poca importancia, por lo que aprenden a base de probar y fallar. El resultado son programadores buenos para codificar usando lenguajes, pero sin bases lógicas sólidas (López, 2011).

Las dificultades anteriores son retos a los que se enfrentan los estudiantes en los cursos de programación, unido a la falta de utilizar buenas prácticas que les permitan desarrollar programas con una solución correcta. Este fenómeno se ha identificado en universidades de México, fundamentándose en los resultados de la presente investigación, después de realizar entrevistas a 55 docentes que imparten clases de programación en diferentes universidades del país, y en los estudios realizados en IES mexicanas; resultados que se exponen en el siguiente apartado de este trabajo.

El objetivo de este estudio de investigación es establecer un conjunto de buenas prácticas, administrativas y de ingeniería, y CC en el proceso de desarrollo de programas para mejorar los cursos de programación, fomentando en los estudiantes su uso y contribuyendo en la mejora de sus programas

Por todo lo anterior, en este artículo se presenta la situación de la enseñanza de la programación en la educación superior en México, así como de las prácticas de ingeniería de *software* que los docentes utilizan. El objetivo de este estudio es establecer un conjunto de buenas prácticas, administrativas y de ingeniería, y CC en el proceso de desarrollo de programas para mejorar los cursos de programación, fomentando en los estudiantes su uso y contribuyendo en la mejora de sus programas.

Para cumplir con el objetivo se llevó a cabo una investigación descriptiva, usando una encuesta (véase anexo al final del artículo) que fue validada con el grado de concordancia entre 7 jueces y con el coeficiente alfa de Cronbach. La encuesta se aplicó a 84 docentes de 38 universidades públicas incorporadas a la Asociación Nacional de Universidades e Instituciones de Educación Superior (ANUIES) de la República Mexicana. Los resultados se obtuvieron mediante un análisis de datos con estadística descriptiva y de componentes principales (CP).

2. Situación de los cursos de programación en universidades de México

En México, las universidades que ofrecen carreras afines con las ciencias computacionales incluyen en sus planes de estudio materias de programación e ingeniería de *software*, que es una de las ocho áreas de conocimiento de los perfiles profesionales de la Asociación Nacional de Instituciones de Educación en Tecnologías de Información (ANIEI). Sin embargo, a nivel nacional, el estatus de la programación en la educación superior no es alentador debido a la dificultad que muestran los estudiantes para entender el problema, al desinterés de seguir una directriz que guíe su desarrollo, a la falta de uso de buenas prácticas y al número de estudiantes que hay en el aula, lo que origina el desarrollo de programas no funcionales y repercute en los índices de suspensos. Lo anteriormente expuesto fue identificado por 55 docentes de 29 universidades mexicanas. En el cuadro 1 se muestran los problemas identificados por los docentes.

Cuadro 1. **Problemas detectados en los estudiantes en función de las entrevistas realizadas a 55 docentes de 29 universidades**

Problema encontrado	Porcentaje (%)
Falta de lógica de algorítmica para solucionar problemas.	85,45
Dificultad en entender y encontrar errores.	83,64
Proceso de codificación a prueba y error (codificación directa).	81,82
Desarrollo de programas que no funcionan correctamente.	81,82
Dificultad en la comprensión de conceptos.	80
Dificultad en la comprensión del problema.	78,18
Falta de uso de buenas prácticas en el desarrollo de programas.	76,36
Cantidad numerosa de estudiantes en el aula (universidades públicas).	74,55
Alto índice de suspensos, ocasionando estrés y descontento en los cursos.	72,73
Desinterés de seguir una directriz que guíe el desarrollo del programa.	69,09

Fuente: elaboración propia.

En este contexto, los estudiantes de las carreras de Ingeniería Electromecánica y de Ingeniería en Sistemas Computacionales del Instituto Tecnológico Superior Progreso (Yucatán), al cursar Fundamentos de Programación, codifican directamente sin entender el problema, enfrentándose con dificultades de seis tipos: fobia a problemas complejos, lógica incompleta, desconocimiento del lenguaje, desconocimiento de herramientas del entorno integrado de desarrollo (IDE), falta de motivación y mala administración del tiempo (Fuentes y Medina, 2017).

En la Universidad Autónoma Indígena de México, en la carrera de Ingeniería en Sistemas Computacionales, a los estudiantes de la materia Programación Orientada a Objetos les resulta difícil la sintaxis del lenguaje, el IDE y el lenguaje de programación en inglés, unido a la escasez de equipos de ordenadores y a la falta de lógica, lo que provoca en ellos frustración y trasciende en la propia institución educativa cuando los egresados se encuentren en el ámbito profesional (Sánchez, Urías y Gutiérrez, 2015).

La Universidad Autónoma Metropolitana (UAM), Unidad Cuajimalpa, publicó en el Informe de Actividades 2013 que entre las materias que presentan altos índices de suspensos se

encuentran Programación, Programación Estructurada y Programación Orientada a Objetos (UAM, 2013). Asimismo, se realizó un estudio de tres IES públicas (Instituto Tecnológico de Chihuahua, Instituto Tecnológico de Ciudad Juárez e Instituto Tecnológico de Mexicali) que indicaba que entre las materias con mayor porcentaje de suspensos estaban Programación, Fundamentos de Programación y Estructura de Datos (Olivares, Jiménez, Ortiz y Rodríguez, 2015).

En la investigación realizada por el Instituto Tecnológico de Ciudad Guzmán, se ha detectado que año tras año las matrículas de las carreras afines a la computación han disminuido, ya que a los alumnos les resulta difícil aprender a programar, aumentando el índice de suspensos y abandono (Michael y Martínez, 2013).

En el área de conocimiento Formación Tecnológica de la Universidad Tecnológica de Jalisco, se imparten materias relacionadas con tecnología que tienen un índice de suspensos del 60 %, y, dentro de esta, las materias de Programación y Desarrollo de *Software* tienen el 70 % de suspensos (Macías, Zamora, Osorio y Jiménez, 2016).

Como se observa en los estudios anteriores, existe el fenómeno de que a los estudiantes les resultan difíciles las materias afines a la programación y, en consecuencia, hay un alto índice de estudiantes no aprobados. Las dificultades para aprender a programar son un fenómeno universal y no específico de un curso, escuela o país (Figueiredo, Gomes y García, 2016). Algunos autores ofrecen resultados que lo evidencian y se refleja en los altos índices de abandono (Machine, 2018). Así, dentro de las dificultades que los estudiantes presentan en los cursos de programación se encuentran la sintaxis, la semántica, la lógica incorrecta, la comprensión de conceptos, el entorno de programación, el razonamiento lógico y la instrucción inadecuada (Bosse y Gerosa, 2016; Insuasti, 2016; Qian y Lehman, 2017). Las deficiencias a las que se enfrentan los estudiantes en los cursos de programación y el alto índice de suspensos repercuten en que los estudiantes desarrollen programas no correctos y defectuosos, siendo algo recurrente en los cursos de programación, por lo que el desarrollo de *softwares* con calidad de código en un ambiente universitario es un desafío (Krusche, Berisha y Bruegee, 2016).

Las dificultades para aprender a programar son un fenómeno universal y no específico de un curso, escuela o país

La baja calidad de los programas desarrollados por los estudiantes se debe a la dificultad de abstracción del problema, a la falta de revisión del diseño y código, a que se centran más en la codificación y a que carecen de habilidades de diseño y de resolución de problemas y no identifican defectos (Nel, Nel y Cronje, 2015). Para ayudar a los estudiantes en la creación de programas con calidad se debe incluir un proceso de desarrollo con buenas prácticas (Rong, Zhang, Qi y Shao, 2016).

En el cuadro 2 se muestran las consecuencias de los problemas identificados, considerando la literatura y a los docentes entrevistados.

Cuadro 2. **Consecuencias de los problemas identificados en los estudiantes de programación según la literatura existente y los docentes entrevistados**

Problema encontrado	Consecuencias
Falta de lógica de algorítmica para solucionar problemas.	Dificultad para crear algoritmos. No se tiene un orden correcto en los pasos. Implementación incorrecta de la solución.
Dificultad en entender y encontrar errores.	No identifican errores y defectos. Inversión de tiempo en corregir sin éxito. Falta de habilidad para usar el depurador.
Proceso de codificación a prueba y error (codificación directa).	Mala administración de tiempo. Lógica incorrecta e incompleta. Desarrollo de programas que no funcionan.
Desarrollo de programas que no funcionan correctamente.	Frustración de los estudiantes. Desmotivación del estudiante. Poca planificación para solucionar el problema.
Dificultad en la comprensión de conceptos.	Desmotivación del estudiante. Desagrado y falta de interés en el curso. Aplicación deficiente de los conceptos.
Dificultad en la comprensión del problema.	Frustración de los estudiantes. Desarrollo de programas no correctos. Abstracción inadecuada de problemas.
Falta del uso de buenas prácticas en el desarrollo de programas.	Falta de revisión en el diseño y en el código. No desarrollan competencias para el campo laboral.
Cantidad numerosa de estudiantes en el aula (universidades públicas).	Falta de equipos de cómputo. Afecta al rendimiento de los alumnos. Desinterés y disgregación del estudiante.
Alto índice de suspensos.	Desagrado y estrés en los cursos de programación. Abandono en programas educativos. Desmotivación del estudiante.
Desinterés de seguir una directriz que guíe el desarrollo del programa.	Se realiza una codificación directa. No desarrollan competencias para el campo laboral.

Fuente: elaboración propia.

3. Prácticas y calidad en el proceso de desarrollo de programas

Los modelos de calidad y estándares, tales como ISO/IEC 9126-1 (ISO/IEC JTC1, 2000), ISO/IEC 25000 (Dave, 2004; Suárez y Garzás, 2014), ISO/IEC 25010:2011 (BS ISO/IEC, 2011), ISO/IEC 25023:2016 (ISO/IEC, 2016), McCall (McCall, Richards y Walters, 1977), Bohem (Boehm, Brown y Lipow, 1976) y Dromey (Dromey, 1995), incluyen CC, también llamados «atributos internos», que permiten evaluar la calidad del proceso y/o del producto en la construcción del *software* en el ámbito empresarial. No obstante, los CC deben estar presentes en los cursos de programación, fomentando en los estudiantes el uso de buenas prácticas en la programación bajo dos perspectivas: guiar el proceso de desarrollo y crear programas que cumplan con los requisitos predefinidos. Es por ello que solo se adaptaron cuatro CC a fin de incorporarlos en el proceso de desarrollo en los cursos de programación: completitud, consistencia, entendible y exactitud, los cuales se seleccionaron en función de la revisión de la literatura y de los resultados de la investigación de este trabajo. Además, en los cursos de programación se deben incluir buenas prácticas, administrativas y de ingeniería, que los estudiantes han de aplicar con la finalidad de mitigar la entrega de programas que no cumplen con los resultados.

Los CC deben estar presentes en los cursos de programación, fomentando en los estudiantes el uso de buenas prácticas en la programación bajo dos perspectivas: guiar el proceso de desarrollo y crear programas que cumplan con los requisitos predefinidos

Las prácticas administrativas son actividades orientadas a la administración de proyectos, que incluyen planeación, coordinación, organización y control, con la finalidad de que el desarrollo del *software* se lleve a cabo de manera organizada

Las prácticas administrativas son actividades orientadas a la administración de proyectos, que incluyen planeación, coordinación, organización y control, con la finalidad de que el desarrollo del *software* se lleve a cabo de manera organizada. Algunos ejemplos de prácticas administrativas son la estimación de tiempo, la estimación de tamaño y el registro de defectos.

Las prácticas de ingeniería se centran en los requisitos, en el análisis, en el diseño y en la construcción del *software* con la finalidad de que el estudiante sea consciente de qué debe hacer y solucione un problema dado de manera correcta. Algunos ejemplos de prácticas de ingeniería son la identificación de requisitos predefinidos, la revisión de diseño y la revisión de código.

Incluir buenas prácticas orienta al estudiante en el proceso de desarrollo en los cursos de programación y le da una perspectiva integral de las actividades que ha de realizar en pro de dar solución a un problema correctamente.

4. Metodología

Para clasificar y describir las prácticas que los docentes utilizan en la enseñanza de la programación y proponer las mejores para el desarrollo de programas en los cursos de programación, se realizó una investigación de tipo descriptiva, utilizando una encuesta que fue aplicada a una muestra por conveniencia de 9 docentes expertos en el área, que realizaron una crítica a la encuesta, orientada hacia la comprensión y el vocabulario.

La validez de la encuesta se realizó mediante el grado de concordancia entre 7 jueces, basándose en la prueba binomial, teniendo como resultado $p = 0{,}042$, que representa un grado de concordancia significativo. Como prueba piloto, se aplicó la encuesta a una muestra por conveniencia de 17 docentes con características similares a la población objetivo. También se validó con el coeficiente alfa de Cronbach, que mide la precisión del constructo, con un resultado de 0,92, que es excelente, ya que los ítems guardan una buena correlación.

La población objetivo fueron 87 IES públicas incorporadas a ANUIES, que ofertan carreras que incluyen en sus planes de estudio materias afines con programación. Se aplicó la encuesta a 84 docentes de 38 universidades, por la poca respuesta para acceder a 66 IES. Con los resultados de la encuesta, y después de aplicar métodos estadísticos, se obtuvieron las mejores prácticas de ingeniería de *software*, incorporándose en el proceso de desarrollo de programas en los cursos de programación, y se alinearon con los CC que fueron adaptados.

5. Resultados

Los resultados derivados de la investigación se dan en dos sentidos:

- Identificar las mejores prácticas utilizadas en la enseñanza de la programación.
- Determinar CC para el proceso de desarrollo de programas.

Ambos, mejores prácticas y CC, son adheridos al proceso de desarrollo, CV, considerando como base el CV en cascada, debido a que sirvió de base para otros modelos y sus fases se adaptan al proceso de desarrollo de programas en el aula.

La finalidad de la investigación es proporcionar a los estudiantes un conjunto de buenas prácticas que guíen el proceso de programación en los cursos de programación para dar solución a problemas correctamente. No obstante, en este trabajo solo se presentan los resultados de la investigación como una propuesta para conjuntar las prácticas de ingeniería y administrativas, así como los CC en un CV, pero no los resultados de la aplicación de la propuesta, debido a que aún se debe realizar.

El CV comprende las fases de requerimientos, análisis, diseño, codificación, compilación, pruebas y mantenimiento. Si bien los requisitos están predefinidos en el planteamiento del problema al que se dará solución por medio de la programación en el ámbito educativo, es importante que los estudiantes conozcan esta fase en las etapas tempranas de su formación profesional. El 58,69 % de los docentes encuestados están a favor de incluir esta fase, ya que se obtiene la compresión del problema, estructurando mejor las ideas para el desarrollo del programa, y el 81 % de docentes indican que es importante que los estudiantes conozcan un CV en las materias de programación.

En el cuadro 3 se muestran las prácticas que los docentes utilizan en el ámbito universitario. Como se puede observar, resulta preocupante que solo el 27 % de los profesores utilicen al menos una práctica.

Es importante mencionar que las respuestas se codificaron con variables dicotómicas para representarlas en una tabla de frecuencias. Los valores perdidos no se consideraron. Posteriormente, se seleccionaron las respuestas de las prácticas de ingeniería de *software* que presentaron una frecuencia de 3 o más, ya que la mayoría tenían una sola respuesta y las que presentaron una frecuencia de 2 se enfocaban al desarrollo de *softwares*, las mismas que se discriminaron, así como las que no pertenecían a la fase; por ejemplo, uso del estándar IEEE 830.

Cuadro 3. **Prácticas utilizadas por los docentes en la enseñanza de la programación**

Fase	Prácticas utilizadas	Frecuencia
Requerimientos	Definición y comprensión del problema.	11
	Catálogo de elementos.	3
Análisis	Identificar atributos, datos, métodos, funciones, interacciones y operaciones.	14
	Casos de uso.	5
	Uso de UML.	4
	Planteamiento de soluciones.	3
Diseño	Modelado UML.	11
	Diagrama de flujo.	12
	Algoritmo.	9
	Lúdicamente.	3
	Pseudocódigo.	3

Fase	Prácticas utilizadas	Frecuencia
▶		
Codificación	Solo lenguaje de programación.	19
	Documentación de código, sangrar e identificadores.	12
	Convertir diagrama de flujo a código	8
	Estándar de codificación y lista de cotejo.	5
Compilación	Entender defectos y corregir.	14
	Conteo de defectos de sintaxis.	5
	Documentar resultados de compilación (bitácora de defectos).	5
	Rastrear errores (uso de *debug*).	5
	Revisión de código antes de compilar.	3
Pruebas	Casos de prueba.	23
	Probar con diferentes datos.	7
Mantenimiento	Mantenimiento correctivo.	15

Fuente: elaboración propia.

Es relevante mencionar que el 13,10 % de los docentes dan importancia a la definición y comprensión del problema, y el 16,67 %, a la identificación de los elementos que conformarán el programa. Estas prácticas se relacionan, ya que, si se entiende el problema, se podrán identificar variables, tipos de datos, operaciones y pasos que hay que seguir, utilizándose en el diseño. Las técnicas de diseño más utilizadas son el modelado UML, los diagramas de flujo y los algoritmos. Los docentes dicen que su uso requiere organizar, clasificar y estructurar los elementos dando semántica al diseño y determinando la lógica del programa.

El 22,70 % de los docentes indican que los estudiantes deben conocer la sintaxis y las reglas de un lenguaje para que el código sea acorde al diseño y el 14,29 % están a favor de un código fácil de entender, el cual debe tener comentarios, sangrado e identificadores, definidos en un estándar de codificación. La detección de defectos es importante y para corregirlos es necesario entender el defecto. El 16,67 % de los docentes aplican está práctica y el 6 % utilizan el *debug*, la bitácora de defectos, y revisan código. El 27,38 % de los docentes señalan que los casos de prueba son relevantes para verificar que los resultados sean correctos y acordes a lo solicitado en los requerimientos para el programa.

El mantenimiento tiene lugar al corregir los defectos que son detectados en las fases previas. El 17,86 % de los docentes opinan que esta fase está implícita en el proceso de desarrollo.

Para seleccionar las mejores prácticas se analizaron las respuestas de 31 variables en escala Likert al realizar un análisis descriptivo y por componentes. Por cada variable, se obtuvo la distribución de frecuencias y la tabla de descriptivos. De acuerdo con los estadísticos descriptivos, el 81 % de las prácticas presentan una media superior a 3, lo que es aceptable, ya que 25 prácticas de 31 son las más usadas por los docentes. Existe una tendencia hacia las prácticas de ingeniería, ya que a los docentes les interesa más que los estudiantes comprendan cómo solucionar un problema que llevar bitácoras de registros, algo que puede resultar tedioso.

En este sentido, se aplicó la medida de adecuación muestral de Kaiser-Meyer-Olkin (KMO) y, al obtener un resultado favorable (véase cuadro 4), se llevó a cabo el análisis de CP, a fin de agrupar las prácticas para la programación y obtener un número menor de variables sin pérdida de información.

Cuadro 4. **Medida de adecuación muestral de KMO**

Medida de adecuación muestral de KMO		0,664
Prueba de esfericidad de Bartlett	Chi-cuadrado aproximado	1.303,933
	gl	465
	Sig.	0,000

Fuente: elaboración propia.

Se obtuvo la matriz de correlación de las 31 variables con un resultado del valor determinante cercano a 0 y la matriz antiimagen con coeficientes bajos. Los valores mayores a 0,7 fueron registro de defectos de sintaxis con defectos lógicos y de ejecución, identificación de datos de entrada con identificación de datos de salida e identificación de fórmulas con identificación de funciones/métodos. Así, los docentes que utilizan estas prácticas también usan las prácticas correlacionadas.

Se obtuvieron las comunalidades de las 31 variables. Las que mejor explican la proporción de la varianza de los componentes comunes oscilan entre 0,804 y 0,894: registrar defectos lógicos, identificar datos de entrada y de salida, uso del diagrama Nassi-Sheiderman, registrar defectos de sintaxis y validación de datos de entrada. Así, en la solución de un problema se requiere del análisis para identificar datos de entrada y salida, del diseño, de la codificación (es deseable llevar un registro de defectos) y de la ejecución para hacer pruebas validando los datos de entrada.

Se obtuvieron 9 componentes comunes que resumen las variables originales, siendo los óptimos para la solución. Los componentes extraídos deben ser aplicados en el proceso de

desarrollo de programas en el aula. El cuadro 5 muestra los componentes (C) que explican el 69,34 % de la varianza de las variables originales.

Cuadro 5. **Porcentaje de la varianza explicada**

Componente	Autovalores iniciales			Suma de las saturaciones al cuadrado de la rotación		
	Total	Porcentaje de la varianza	Porcentaje acumulado	Total	Porcentaje de la varianza	Porcentaje acumulado
1	6,965	22,466	22,466	4,082	13,167	13,167
2	3,378	10,897	33,363	3,782	12,201	25,368
3	2,240	7,226	40,590	2,213	7,140	32,508
4	1,785	5,758	46,348	2,156	6,955	39,463
5	1,683	5,430	51,778	2,028	6,542	46,005
6	1,580	5,098	56,876	1,946	6,278	52,284
7	1,476	4,763	61,638	1,862	6,006	58,290
8	1,250	4,031	65,669	1,793	5,785	64,074
9	1,139	3,673	69,342	1,633	5,268	69,342
10	0,982	3,168	72,510			
11	0,920	2,969	75,479			
…						
31	0,034	0,108	100			

Fuente: elaboración propia.

Con la correlación entre las variables y los componentes se determinó el nombre de cada componente. En el cuadro 6 se aprecia que el componente 1 (análisis de requisitos) se describe por la identificación de funciones/métodos (A), identificación de fórmulas (B), prueba de validación de datos (C), programa claro y legible (D) e identificación de datos de entrada

y de salida (E). Es decir, a mayor identificación de las variables, mejor será el análisis de requisitos y, a mayor verificación de los resultados correctos, menor será el análisis realizado.

Cuadro 6. **Matriz de los componentes extraídos**

	Componente								
	1	2	3	4	5	6	7	8	9
A	0,711	– 0,200	– 0,224	– 0,206	– 0,046	– 0,087	– 0,258	0,253	0,074
B	0,665	– 0,302	0,001	– 0,310	– 0,027	– 0,003	– 0,022	0,331	0,135
C	0,656	0,035	– 0,287	0,051	0,142	0,218	0,148	– 0,102	0,115
D	0,613	0,015	– 0,233	0,216	0,063	– 0,127	– 0,099	– 0,214	– 0,266
E	0,606	– 0,596	– 0,141	– 0,201	0,127	– 0,151	0,036	0,040	– 0,086
F	0,594	0,117	– 0,065	0,131	0,099	– 0,305	0,042	– 0,134	0,234
G	0,573	0,049	– 0,172	0,059	– 0,350	– 0,044	– 0,171	– 0,142	– 0,080
H	0,543	– 0,267	– 0,016	– 0,172	– 0,325	– 0,029	0,204	0,107	– 0,040
I	0,534	0,085	0,181	0,213	– 0,196	– 0,224	0,003	– 0,085	– 0,446
J	0,499	– 0,136	0,071	0,351	– 0,212	0,260	0,088	– 0,128	– 0,468
K	0,481	– 0,199	– 0,052	– 0,357	– 0,327	0,187	– 0,064	– 0,212	0,159
L	0,474	0,091	0,113	0,450	– 0,247	0,023	0,030	0,029	0,081
M	0,466	– 0,217	– 0,024	0,079	– 0,187	0,216	– 0,440	– 0,422	0,112
N	0,442	0,692	– 0,173	– 0,223	0,059	– 0,140	0,191	– 0,011	– 0,052
O	0,544	0,645	– 0,148	– 0,293	0,170	– 0,115	0,162	– 0,033	– 0,072
P	0,603	– 0,619	– 0,113	– 0,195	0,109	– 0,174	0,122	0,056	0,053
Q	0,568	0,591	– 0,130	– 0,213	0,185	0,040	0,076	– 0,024	– 0,113

	Componente								
	1	2	3	4	5	6	7	8	9
▶									
R	0,434	0,534	– 0,109	0,002	0,223	– 0,151	– 0,068	– 0,013	0,025
S	0,335	– 0,533	0,040	0,061	0,403	0,069	– 0,046	– 0,172	– 0,061
T	0,252	– 0,240	0,654	– 0,013	0,049	– 0,280	0,040	0,170	– 0,044
U	0,193	0,053	0,632	– 0,243	– 0,110	0,300	0,234	0,159	– 0,185
V	0,302	0,074	0,592	– 0,411	0,001	0,115	– 0,043	– 0,285	0,190
W	0,326	0,005	0,474	– 0,008	0,332	– 0,156	– 0,238	0,210	– 0,313
X	0,388	0,249	0,176	0,413	0,190	0,132	– 0,339	0,270	0,111
Y	0,314	0,173	0,174	0,276	– 0,516	– 0,298	0,175	0,031	0,276
Z	0,255	– 0,335	0,173	0,292	0,414	0,021	0,290	– 0,296	0,212
AA	0,357	0,427	0,402	– 0,111	– 0,027	0,539	– 0,150	– 0,122	0,130
BB	0,359	– 0,061	0,008	0,293	0,279	0,321	0,628	0,012	0,105
CC	0,362	0,014	– 0,174	0,173	0,260	0,342	– 0,424	0,333	0,089
DD	0,252	– 0,036	– 0,278	0,111	– 0,278	0,401	0,257	0,448	– 0,006
EE	0,347	0,168	0,296	0,280	– 0,059	– 0,337	– 0,032	0,138	0,363

Nota: A (identificar funciones y métodos), B (identificar fórmulas), C (prueba de validación de datos), D (verificar que el programa sea claro y legible), E (identificar datos de salida), F (depurar código), G (uso de comentarios en el código), H (identificar variables que se van a utilizar), I (hacer seguimiento de variables), J (operar programas), K (conocer la sintaxis del lenguaje), L (trascribir código), M (codificar de manera modular), N (registrar defectos de sintaxis), O (registrar defectos lógicos), P (identificar datos de entrada), Q (registrar defectos en ejecución), R (registrar tiempo de desarrollo), S (ejecución correcta del programa), T (uso de algoritmo), U (uso de diagrama de flujo), V (codificar de manera secuencial), W (uso de pseudocódigo), X (uso de diagrama de actividades), Y (modularizar programa lineal), Z (verificar resultados correctos), AA (uso de diagrama Nassi-Scheiderman), BB (validación de datos de entrada), CC (uso de diagrama de clases), DD (revisión de código [defectos de sintaxis]) y EE (uso de prueba de escritorio).

Fuente: elaboración propia.

Con la finalidad de facilitar la interpretación de los componentes resultantes y que cada componente no esté saturado en más de un factor, se obtuvo la matriz de componentes

rotados con el método varimax, considerando para cada componente las variables con una saturación > 0,400, reduciendo así las 31 variables en 9 componentes. La variable uso de pseudocódigo se excluye, ya que no cumple con la saturación mínima (véase cuadro 7, donde dicha variable no aparece sombreada en ninguno de sus valores). Así, en el desarrollo de programas en los cursos de programación se deben considerar los 9 componentes: 1 (análisis de requisitos), 2 (bitácoras y prueba), 3 (diseño en programación estructurada), 4 (seguimiento del código), 5 (verificación), 6 (comprobación del programa), 7 (diseño en programación orientada a objetos), 8 (codificación) y 9 (revisión de código).

Cuadro 7. **Matriz de componentes rotados**

	Componente								
	1	2	3	4	5	6	7	8	9
P	0,869	– 0,030	– 0,029	0,097	0,030	0,295	– 0,046	0,050	– 0,030
E	0,851	– 0,008	– 0,043	0,198	– 0,073	0,234	– 0,001	0,054	– 0,066
B	0,787	0,133	0,209	– 0,019	0,121	0,007	0,184	0,080	0,149
A	0,746	0,233	– 0,058	0,098	0,114	– 0,119	0,302	0,192	0,064
H	0,574	0,058	0,161	0,233	0,184	– 0,002	– 0,165	0,122	0,261
O	0,098	0,931	0,104	0,061	0,047	– 0,019	– 0,004	0,002	0,020
N	– 0,003	0,883	0,047	0,056	0,120	– 0,085	– 0,056	– 0,020	0,084
Q	0,075	0,849	0,134	0,131	– 0,019	0,015	0,132	0,068	0,067
R	– 0,006	0,683	– 0,048	0,069	0,159	0,019	0,229	0,011	– 0,107
C	0,316	0,423	– 0,080	0,141	0,031	0,386	0,168	0,284	0,255
U	0,045	0,011	0,814	0,127	– 0,002	0,009	– 0,041	– 0,091	0,152
V	0,112	0,158	0,688	– 0,118	0,098	0,085	– 0,078	0,347	– 0,285
AA	– 0,207	0,321	0,613	0,012	0,043	0,045	0,321	0,435	0,107
T	0,301	– 0,174	0,494	0,139	0,320	0,090	0,039	– 0,268	– 0,308

	Componente								
	1	2	3	4	5	6	7	8	9
▶									
W	0,206	0,103	0,393	0,275	0,005	0,001	0,379	– 0,295	– 0,389
J	0,106	– 0,019	0,140	0,770	0,010	0,196	0,085	0,157	0,238
I	0,160	0,215	0,133	0,709	0,248	– 0,033	– 0,001	– 0,026	– 0,099
D	0,264	0,339	– 0,226	0,545	0,060	0,167	0,142	0,174	– 0,093
G	0,278	0,238	– 0,089	0,411	0,229	– 0,133	0,051	0,391	0,084
Y	0,024	0,076	0,044	0,155	0,780	– 0,070	– 0,151	0,089	0,145
EE	0,075	0,130	0,086	– 0,014	0,696	0,076	0,201	– 0,053	– 0,138
L	0,030	0,072	0,040	0,366	0,503	0,143	0,208	0,146	0,194
F	0,281	0,411	– 0,127	0,106	0,423	0,255	0,082	0,132	– 0,148
Z	0,126	– 0,090	0,029	0,036	0,112	0,785	0,015	0,033	– 0,161
BB	0,057	0,164	0,122	0,089	0,071	0,746	0,023	– 0,113	0,420
S	0,419	– 0,161	0,003	0,163	– 0,205	0,497	0,170	0,087	– 0,225
CC	0,186	0,100	– 0,077	0,007	– 0,088	0,035	0,759	0,123	0,157
X	– 0,057	0,159	0,090	0,144	0,293	0,080	0,720	0,004	– 0,001
M	0,217	– 0,069	– 0,011	0,269	0,064	0,077	0,214	0,734	– 0,134
K	0,453	0,087	0,187	0,037	0,032	– 0,043	– 0,129	0,576	0,129
DD	0,191	0,018	– 0,008	0,096	0,035	– 0,016	0,146	– 0,026	0,768

Nota: véase nota situada al pie del cuadro 6.

Fuente: elaboración propia.

Para identificar los CC se consideró la revisión de la literatura de los modelos de calidad ya mencionados. Basándose en la definición de cada característica, de su clasificación en el modelo y de su adaptación en el proceso de desarrollo, se seleccionaron los CC (véase cuadro 8).

Cuadro 8. **Criterios de calidad para el desarrollo de programas en el aula**

Criterio	Descripción
Completitud	Grado en que el programa cumple con todos los requisitos predefinidos y la funcionalidad requerida.
Consistencia	Grado en que se utiliza un diseño uniforme que tenga trazabilidad con la codificación.
Entendible	Grado en que un código es entendido.
Exactitud	Capacidad que tiene el programa para proporcionar los resultados correctos.

Fuente: elaboración propia.

Con la finalidad de asociar los CC con las prácticas, se consideraron 25 variables seleccionando las favorables con el método de escalamiento tipo Likert (Sampieri, Fernández y Baptista, 2006), donde, a partir de un conjunto de ítems con un valor asignado, se obtienen las puntuaciones por variable. Se consideró el valor de 2,5 como la puntuación mínima favorable. Como se muestra en el cuadro 9, las puntuaciones sombreadas en gris oscuro son las propicias.

Los valores asociados a la escala Likert del cuadro 9 son 5 (siempre), 4 (casi siempre), 3 (mayoría de veces), 2 (pocas veces) y 1 (nunca).

Cuadro 9. **Selección de puntuaciones**

Componente						
	5	4	3	2	1	Puntuación
Cuando realiza la verificación del aprendizaje, usted comprueba...						
Que los resultados sean correctos	310	44	21	4	2	3,69
Validación de los datos de entrada	195	108	21	20	1	2,32

▶

Componente						
	5	4	3	2	1	Puntuación
▶						
Ejecución correcta del programa	285	76	6	2	5	3,39
La estrategia de aprendizaje	175	96	39	18	2	2,11*
Cuando los alumnos desarrollan un programa, usted solicita...						
Registrar defectos de sintaxis	55	32	42	50	26	0,65
Registrar defectos lógicos	45	76	45	30	26	0,54
Registrar defectos en ejecución	35	68	54	38	23	0,42
Registrar tiempo de desarrollo	80	36	45	38	25	0,95
Verificar resultados correctos	290	64	12	4	4	3,45
Verificar programa claro y legible	215	100	18	12	4	2,56
Poner comentarios en el código	160	100	30	10	12	1,90
Cuando se les plantea a los alumnos un problema, deben analizarlo para...						
Identificar variables que van a utilizar	255	88	21	4	2	3,04
Identificar los datos de entrada	315	68	3	0	3	3,75
Identificar los datos de salida	310	64	3	2	4	3,69
Identificar fórmulas si existen	240	92	18	6	4	2,86
Identificar funciones o métodos	270	80	15	2	4	3,21
Los alumnos, previo a la codificación, deben diseñar, usando...						
Un algoritmo	225	84	24	8	6	2,68
Un pseudocódigo	160	100	39	12	8	1,90
Un diagrama de flujo	195	76	33	10	10	2,32

▶

Componente						
	5	4	3	2	1	Puntuación
▶						
En la codificación, los alumnos deben...						
Poner comentarios	225	84	18	14	5	2,68
Conocer la sintaxis del lenguaje	250	76	27	4	4	2,98
Codificar de manera secuencial	145	68	48	24	10	1,73
Codificar de manera modular	200	72	45	10	6	2,38
Las pruebas que se realizan a un programa son para...						
Corroborar resultados correctos	320	60	6	0	3	3,81
Corroborar datos de entrada	225	88	21	10	5	2,68

* Para el resultado se realizó la división por 83 debido a valor perdido.

Fuente: elaboración propia.

Con los resultados obtenidos del modelo de CP, del escalamiento tipo Likert y de los CC, se incluyeron las prácticas de ingeniería y administrativas (complementarias), así como los CC en las fases del CV (véase cuadro 10), a excepción de la fase de mantenimiento que fue eliminada por no aportar valor con ninguna práctica.

Cuadro 10. **Prácticas y criterios de calidad en el CV para la programación**

Fase	Prácticas	CC
Requerimientos	Comprender el problema.	Sin criterio
Análisis	Identificar variables que se van a utilizar. Identificar los datos de entrada. Identificar los datos de salida. Identificar fórmulas. Identificar funciones o métodos.	Completitud
Complementaria	Registrar tiempo.	

▶

Fase	Prácticas	CC
▶		
Diseño	Diagrama de flujo. Diagrama de Nassi-Scheiderman. Algoritmo. Diagrama de clases. Diagrama de actividades.	Consistencia
Complementarias	Diseñar prueba de validación. Pruebas de escritorio. Registrar tiempo.	
Codificación	Poner comentarios. Conocer la sintaxis del lenguaje. Verificar programa claro y legible. Codificar de manera secuencial. Codificar de manera modular. Revisión de código.	Entendible
Complementarias	Operar programas. Seguimiento de variables. Depurar código. Transcribir código. Modularizar un programa lineal. Registro de tiempo.	
Compilación	Registra defectos lógicos. Registrar defectos de sintaxis. Registrar defectos en ejecución. Registrar tiempo.	Sin criterio
Pruebas	Verificar resultados correctos. Validación de datos de entrada. Ejecución correcta del programa.	Exactitud
Complementaria	Registrar tiempo de desarrollo.	

Fuente: elaboración propia.

Finalmente, se contrastaron las prácticas obtenidas con las de los docentes, concordando que en el análisis se identifican los elementos; en el diseño se usan diagramas de flujo, algoritmo y modelado UML; en la codificación se usan comentarios, conocimiento de la sintaxis y revisión de código; en la compilación se incluyen bitácoras de defectos; y en las pruebas se verifican resultados.

6. Conclusiones

En este estudio se muestra el nivel de los cursos de programación en las universidades mexicanas. Se ha detectado que los estudiantes tienen dificultades en los cursos relacionados con la programación y que, a la hora de desarrollar un programa, les hace falta experiencia y habilidad del uso de buenas prácticas de ingeniería de *software* y administrativas que les permitan crear programas de calidad

El propósito de la investigación fue establecer un conjunto de buenas prácticas y CC en el proceso de desarrollo de programas en los cursos de programación en el ámbito universitario, contribuyendo en la formación profesional de los estudiantes.

Se identificaron 23 prácticas que utilizan los docentes en los cursos de programación. Sin embargo, un 33,14 % de docentes no indicaron el uso de alguna práctica y son pocos los que las utilizan en el aula, lo que resulta preocupante, dado que el docente debe favorecer y fomentar el uso de estas en el aula.

El análisis de CP permitió agrupar las prácticas para la programación en 9 componentes. Los componentes 1 y del 3 al 9 son prácticas de ingeniería y el componente 2 son prácticas administrativas. Por ende, se pueden conocer las prácticas que los estudiantes deben emplear para que su proceso de desarrollo contribuya en la creación de programas funcionales.

Se adaptaron 4 CC: completitud, consistencia, entendible y exactitud. La importancia de incluirlos en el CV se debe a que los estudiantes se limitan a dar solución a un problema sin considerar un análisis exhaustivo, un diseño coherente con el análisis que sirva de base en la codificación, un código entendible y resultados correctos. El estándar ISO 29110 debe tenerse en cuenta en trabajos futuros para conocer la madurez del proceso de desarrollo de los estudiantes. Asimismo, se deberán aplicar las prácticas y CC en el CV para el desarrollo de programas en un curso de programación con la finalidad de obtener los resultados del efecto que se tiene con los estudiantes.

Las prácticas adheridas al CV no son determinantes para desarrollar *software* de calidad a nivel empresarial, pero sirven de apoyo para que los estudiantes desarrollen programas correctos y sean conscientes de que el uso de buenas prácticas mejora la calidad en el desarrollo. Lo anterior impactará en su formación profesional y, como egresados, podrán adaptarse y desarrollarse en el mercado laboral

En conclusión, el proceso de desarrollo en las aulas debe contar con prácticas de ingeniería y administrativas alineadas a CC (véase cuadro 10) para que los estudiantes desarrollen programas con calidad y se habitúen al uso de buenas prácticas, potenciando sus habilidades. Las prácticas adheridas al CV no son determinantes para desarrollar *software* de calidad a nivel empresarial, pero sirven de apoyo para que los estudiantes desarrollen programas correctos y sean conscientes de que el uso de buenas prácticas mejora la calidad en el desarrollo. Lo anterior impactará en su formación profesional y, como egresados, podrán adaptarse y desarrollarse en el mercado laboral.

Referencias bibliográficas

Boehm, B. W., Brown, J. R. y Lipow, M. (1976). Quantitative evaluation of software quality. *ICSE '76 Proceedings of the 2nd International Conference on Software Engineering*, 592-605.

Bosse, Y. y Gerosa, M. A. (2016). Why is programming so difficult to learn? Patterns of difficulties related to programming learning. *ACM SIGSOFT Software Engineering Notes*, *41*(6), 1-6. doi: 10.1145/3011286.3011301.

Bosse, Y. y Gerosa, M. A. (2017). Difficulties of programming learning from the point of view of students and instructors. *IEEE Latin America Transactions*, *5*(11), 2.191-2.199. doi: 10.1109/TLA.2017.8070426.

BS ISO/IEC. (2011). Systems and software engineering-Systems and software quality requirements and evaluation (SQUARE)-System and software quality models. *BSI Standards Publication*. Recuperado de <https://pdfs.semanticscholar.org/57a5/b99eceff9da205e244337c9f4678b5b23d25.pdf> (consultado el 9 de diciembre de 2016).

Dave, Z. (2004). *Software Quality Requirements and Evaluation, the ISO 25000 Series.* Recuperado de <http://www.psmsc.com/downloads/twgfeb04/04zubrowiso25000swqualitymeasurement.pdf> (consultado el 9 de diciembre de 2019).

Dromey, R. G. (1995). A model for software product quality. *IEEE Transactions on Software Engineering*, *21*(2), 146-162. doi: 10.1109/32.345830.

Figueiredo, J., Gomes, N. y García, F. J. (2016). ne-Course for learning programming. *Proceedings of the 4th International Conference on Technological Ecosystems for Enhancing Multiculturality*, 549-553. doi: 10.1145/3012430.3012572.

Fuentes, J. y Medina, M. (2017). Dificultades de aprender a programar. *Educación en Ingeniería*, *12*(24), 76-82. doi: 10.26507/rei.v12n24.728.

Insuasti, J. (2016). Problemas de enseñanza y aprendizaje de los fundamentos de programación. *Educación y Desarrollo Social*, *10*(2), 234-246. doi: org/10/18359/reds.1701.

ISO/IEC. (2016). Systems and software engineering-Systems and software quality requirements and evaluation (SQUARE)-Measurement of system and software product quality. *ISO/IEC*. Recuperado de <https://www.sis.se/api/document/preview/920620/> (consultado el 6 de diciembre de 2019).

ISO/IEC JTC1. (2000). Information technology-Software product quality. Part 1. Quality model. *ISO/IEC*. Recuperado de <https://www.cse.unsw.edu.au/~cs3710/PMmaterials/Resources/9126-1%20Standard.pdf> (consultado el 6 de diciembre de 2019).

Krusche, S., Berisha, M. y Bruegee, B. (2016). Teaching code review management using branch based workflows. *IEEE/ACM. 38th IEEE International Conference on Software Engineering Companion*, 384-393. doi: 10.1145/2889160.2889191.

López, L. (2011). Metodologías para la enseñanza del aprendizaje de la programación estructurada y orientada a objetos. *Sistemas, Cibernética e Informática*, *8*(1), 52-60.

Machine, I. (2018). Intelligent tutor for programming system using multiple intelligences. *IEEE Latin America Transactions*, *16*(2), 634-638. doi: 10.1109/TLA.2018.8327423.

Macías, H. R., Zamora, V. M., Osorio, S. y Jiménez, M. (2016). The development skills in software design using a virtual learning environment. *Tecnología Educativa. Revista CONAIC*, *3*(1), 22-28.

McCall, J. A., Richards, P. K. y Walters, G. F. (1977). *Factors in Software Quality*. RADC TR-77-369. Rome: Rome Air Development Center. Recuperado de <https://pdfs.semanticscholar.org/82a9/18fd83f1c0addb890ef313ff892807a10a11.pdf> (consultado el 7 de diciembre de 2019).

Michael, N. y Martínez, P. (2013). Mejora del aprendizaje de Programación Orientada a Objetos. *Revista Iberoamericana para la Investigación y el Desarrollo Educativo*, *10*.

Nel, G., Nel, L. y Cronje, J. (2015). Attributes contributing to students´ use of quality software development practices. *The African Journal of Information and Communications*, *15*(1), 38-50.

Olier, A. J., Gómez, A. A. y Caro, M. F. (2017). Design and implementation of a teaching tool for introduction to object oriented programming. *IEEE Latin America Transactions*, *15*(1), 97-102. doi: 10.1109/TLA.2017.7827913.

Olivares, J. C., Jiménez, J. A., Ortiz, O. y Rodríguez, N. E. (2015). Desarrollo de una aplicación para fortalecer el aprendizaje de los fundamentos de programación. *Revista de Ciencia e Ingeniería del Instituto Superior de Coatzacoalcos*, *2*(2), 1-4.

Ozmen, A. y Altun, A. (2014). Undergraduate students experiences in programming: difficulties and obstacles. *Turkish Online Journal of Qualitative Inquiry*, *5*(4), 9-27.

Qian, Y. y Lehman, J. (2017). Students´ misconceptions and other difficulties in introductory programming: a literature review. *ACM Transations on Computing Education*, *18*(1), 1-24. doi: 10.1145/3077618.

Romero, M., Lepage, A. y Lille, B. (2017). Computational thinking development through creative programming in higher education. *International Journal of Educational Technology in Higher Education*, *14*(1), 1-15.

Rong, G., Zhang, H., Qi, S. y Shao, D. (2016). Can software engineering students program defect-free? An educational approach. *IEEE/ACM. 38th IEEE International Conference on Software Engineering Companion*, 364-373. doi: 10.1145/2889160.2889189.

Sampieri, R., Fernández, C. y Baptista, B. (2006). *Metodología de la investigación*. México: McGraw-Hill.

Sánchez, J., Urías, M. y Gutiérrez, B. (2015). Análisis de los problemas de aprendizaje de la programación orientada a objetos. *Ra Ximhai*, *11*(4), 289-304.

Suárez, L. F. y Garzás, P. J. (2014). *I Jornada sobre calidad del producto software e ISO 25000.* Santiago de Compostela, España: Xunta de Galicia.

UAM. (2013). *Informe de actividades 2013*. Recuperado de <http://www.cua.uam.mx/pdf/Informe_anual_2013_Unidad_Cuajimalpa_Rector.pdf> (consultado el 19 de diciembre de 2019).

Anexo

Plantilla de la encuesta

INSTRUCCIONES. Estimado profesor, por favor, lea cuidadosamente cada una de las preguntas que se presentan en el instrumento. Se solicita su valiosa colaboración para responder a todas y cada una de las preguntas con veracidad, agradeciendo de antemano su franqueza. La información obtenida será utilizada para fines de investigación. Gracias por su apoyo.

1. Entidad federativa de la universidad de procedencia

.....

2. Escuela o universidad de procedencia

.....

3. ¿Cuáles de las siguientes materias ha impartido usted en licenciaturas donde se cursan asignaturas relacionadas con la programación?

- ○ Algoritmos Computacionales
- ○ Programación Estructurada
- ○ Estructura de Datos
- ○ Programación Avanzada
- ○ Programación Orientada a Objetos
- ○ Otra. Especifique cuál: .....

4. Cuando imparte materias relacionadas con programación, ¿usted sigue algún proceso para el desarrollo de los programas?

- ○ Sí
- ○ No

Si su respuesta es «Sí», explique en qué consiste:

.....

.....

5. Elija el enfoque con el que usted trabaja cuando imparte las materias relacionadas con la programación

- ○ Uso de lenguaje de programación
- ○ Uso de lenguaje algorítmico

▶

▶ (*cont.*)

6. ¿Con cuál(es) de las siguientes herramientas se apoya para la enseñanza de la programación?

○ Scratch ○ Alice ○ Logo ○ Ninguna

○ Small Basic ○ Kodu ○ KPL ○ Otra. Especifique cuál: .....

7. Considera que los alumnos deben conocer lo que es un ciclo de vida del *software* para las materias de programación

○ Totalmente de acuerdo ○ De acuerdo ○ Ni de acuerdo ni en desacuerdo

○ En desacuerdo ○ Totalmente en desacuerdo

8. Considera que la fase de requerimientos es importante en la enseñanza de la programación

○ Sí

○ No

¿Por qué?

.....

9. Por cada una de las fases presentadas, indique cuáles son las mejores prácticas que utiliza con sus alumnos en la enseñanza de la programación

Fase	Mejores prácticas que usted utiliza
Requerimientos	
Análisis	
Diseño	
Codificación	
Compilación	
Pruebas	
Mantenimiento	

▶

▶ (*cont.*)

10. Enumere del 1 al 9 las habilidades específicas que considera que los alumnos deben tener para la programación (1 es la de mayor prioridad)

Razonamiento lógico-matemático	☐	Pensamiento algorítmico	☐
Uso de herramientas de diagramación	☐	Codificar una solución en LP	☐
Uso de herramientas de los LP[1]	☐	Detección de defectos en el programa	☐
Reglas sintácticas y semánticas del LP	☐	Uso de técnicas de diseño	☐
Uso de la técnica «divide y vencerás»	☐		

Si hay otra(s) que considere, menciónela(s): .....

11. Enumere del 1 al 9 las habilidades genéricas que considera que los alumnos deben tener para la programación (1 es la de mayor prioridad)

Trabajo en equipo	☐	Analizar información	☐
Interpretar información	☐	Extraer conclusiones	☐
Organización de información	☐	Clasificar información	☐
Capacidad de comunicación	☐	Adaptación a cambios	☐
Toma de decisiones	☐		

Si hay otra(s) que considere, menciónela(s): .....

12. Para cada una de las siguientes opciones, marque con una X, según corresponda: 1 (siempre), 2 (casi siempre), 3 (la mayoría de las veces), 4 (pocas veces) y 5 (nunca)

Cuando realiza la verificación del aprendizaje, usted comprueba...	1	2	3	4	5
Que los resultados sean correctos					
La validación de los datos de entrada					
Que la ejecución del programa sea correcta					
La estrategia de aprendizaje					

▶

▶

▶ (*cont.*)

Cuando los alumnos desarrollan un programa, usted solicita...	1	2	3	4	5
Registrar defectos de sintaxis					
Registrar defectos lógicos					
Registrar defectos en ejecución					
Registrar tiempo de desarrollo					
Verificar resultados correctos					
Verificar que el programa sea claro y legible					
Poner comentarios en el código					
Cuando se les plantea a los alumnos un problema, deben analizarlo para...	**1**	**2**	**3**	**4**	**5**
Identificar las variables que hay que utilizar					
Identificar los datos de entrada					
Identificar los datos de salida					
Identificar fórmulas, si existen					
Identificar funciones o métodos					
Los alumnos, previo a la codificación, deben diseñar, usando...	**1**	**2**	**3**	**4**	**5**
Un algoritmo					
Un pseudocódigo					
Un diagrama de flujo					
En la codificación, los alumnos deben...	**1**	**2**	**3**	**4**	**5**
Poner comentarios					
Conocer la sintaxis del lenguaje					
Codificar de manera secuencial					
Codificar de manera modular					

▶

▶

▶ (*cont.*)

Las pruebas que se le realizan a un programa son para...	1	2	3	4	5
Corroborar resultados correctos					
Validación de los datos de entrada					
Las actividades que incluye en su didáctica son...	**1**	**2**	**3**	**4**	**5**
Revisión de código					
Depurar código					
Transcribir código					
Realizar pruebas de escritorio					
Modularizar un programa lineal					
Operar los programas					
Hacer seguimiento de variables					
Uso de diagramas de Nassi-Scheiderman					
Uso de diagramas de clases					
Uso de diagramas de actividades					

Gracias por su colaboración.

[1] Lenguaje de programación.

Grados oficiales

¡Inicio en octubre y febrero de cada año!

- 100% online
- 240 créditos
- Bolsa de trabajo
- Descuentos a la excelencia académica

Ingeniería de Tecnologías y Servicios de Telecomunicación

El papel estratégico del sector de las TIC, y la aplicación creciente de estas en los distintos sectores de la sociedad, ha hecho aumentar la necesidad de profesionales de la telecomunicación, una demanda que crecerá exponencialmente en los próximos años. Este grado habilita para el ejercicio de la profesión de ingeniero técnico de telecomunicación, otorgando las competencias necesarias para conseguir las atribuciones profesionales de un ingeniero técnico de telecomunicación y ofreciendo una formación que capacita al estudiante a la hora de analizar, diseñar, implementar, explotar y gestionar sistemas, componentes y procesos del ámbito de las TIC.

Psicología (Rama CC. de la Salud)

Siguiendo el modelo científico-profesional de psicólogo (o *scientist-practitioner*), se trata de aportar a los alumnos los conocimientos científicos necesarios para comprender, interpretar, analizar y explicar el comportamiento humano, así como para evaluar e intervenir en el ámbito individual y social, con el fin de que los psicólogos y la psicología promuevan y mejoren la salud y la calidad de vida de las personas.

Historia

Se conjugan los conocimientos humanísticos básicos y generalistas con el aprendizaje de las herramientas y técnicas de las nuevas TIC. Los estudiantes adquirirán la formación, los conocimientos y las habilidades necesarias para permitirles el pleno desarrollo de las funciones relacionadas con la investigación y la enseñanza de la historia, con la finalidad de que comprendan y hagan comprensibles a los demás los acontecimientos del pasado.

Máster en **Psicopedagogía**

Este máster oficial [60 créditos ECTS] tiene una duración normal de 12 meses.

Dirigido a: Personas vinculadas con el mundo de la educación formal y no formal que deseen actualizar sus conocimientos. El estudiante de este máster ha de estar interesado por la labor del educador en un enfoque educativo inclusivo en el ámbito de la educación formal, y en el trabajo con diferentes grupos o colectivos sociales, favoreciendo la mejora de sus condiciones de vida y la disminución de las desigualdades por motivos de carácter social y cultural.

Objetivos: Permite el desempeño de una labor profesional especializada, avanzada y focalizada en el análisis, la planificación y la intervención para la mejora de los contextos educativos, sociolaborales y sociocomunitarios, de ahí la necesidad de una formación de posgrado que permita el desarrollo de las competencias específicas y multidisciplinares requeridas para su práctica profesional. Pretende dar cobertura a las funciones básicas de los psicopedagogos en distintos ámbitos.

Inicio en **octubre** y **febrero** de cada año **www.udima.es | 918 561 699**

Publicaciones de interés

Área de Tecnología, Ciencia y Educación

Historia del pensamiento contemporáneo

Juan Padilla Moreno

El objeto de este manual es la evolución del pensamiento en los dos últimos siglos de la historia occidental, desde finales del siglo XVIII hasta la actualidad. Siendo el tramo más reciente de una historia milenaria iniciada en la Grecia antigua, es también el más denso y complicado. A dos épocas claramente diferenciadas en el siglo XIX, el Romanticismo y la época positivista, sucede una gran crisis a comienzos del XX, que afecta no solo a la filosofía, sino también a la ciencia, la política, el arte o la religión. Desde entonces la historia del pensamiento se hace más fragmentaria y tanto más problemática cuanto más se aproxima a la actualidad.

En esta obra, el autor prima ante todo la claridad, especialmente necesaria por tratarse de un pensamiento que, dada su larga historia, se ha hecho sumamente sutil y complejo, pero procurando ser riguroso y fiel a las fuentes.

Más información en **tienda.cef.udima.es | 914 444 920**