ISSN: 2448 - 6574

Estimación de errores metodológicos en los cursos de programación: hacia una evaluación crítica de la enseñanza del desarrollo de software

José Luis López Goytia
jlgoytia@gmail.com

Víctor Garduño Mendieta
vicgardm@yahoo.com.mx

Mario Oviedo Galdeano
mog974@yahoo.com.mx

IPN-UPIICSA

Área temática: Evaluación del aprendizaje y del desempeño escolar

Resumen

En el siguiente trabajo se pretende exponer una serie de ejercicios aplicados al iniciar la unidad de aprendizaje de Algoritmos Computacionales. En particular, sobre el repaso de temas de Lógica de Programación, el primer curso de programación para las carreras de informática de la Unidad Profesional Interdisciplinaria de Ingeniería y Ciencias Sociales y Administrativas (UPIICSA) del Instituto Politécnico Nacional (IPN).

Específicamente se tratará de estimar el grado de errores metodológicos cometidos en el desarrollo de los programas. En la práctica, constituye una evaluación en este tema de los cursos de Lógica de Programación. Se pretende argumentar que constituye un problema igual de importante que la parte de codificación y al cual no se le suele poner atención.

Palabras clave: Errores metodológicos, enseñanza de la programación, evaluación en la programación.

ISSN: 2448 - 6574

Planteamiento del problema

En la UPIICSA, del IPN, los profesores de la Academias de Computación y la Academia de Informática de la UPIICSA suelen referirse desde hace varios años a una situación delicada con una frase genérica: “los alumnos no saben programar”. La aplican a las dos carreras de informática de la escuela: Lic. en Ciencias de la Informática e Ing. en Informática. Pero “no saber programar” tiene muchas vertientes. Una de ellas, poco explorada en los cursos de programación, son los errores metodológicos: aquellos que no tienen que ver directamente con la codificación en sí, pero son indispensables en la construcción de un software de calidad.

El problema no parece ser menor. López y Gutiérrez (2014: 237) expresan que entre 2009 y 2011 se aplicaron aproximadamente 200 encuestas en un curso intermedio de base de datos. Los estudiantes identificaron entre 38% y 43% de errores típicos al no haber seguido recomendaciones básicas de la metodología más sencilla para desarrollo de software (método de cascada). Los errores se repartieron en todas las etapas (autorización del proyecto, análisis, diseño, codificación y pruebas modulares, pruebas integrales y mantenimiento) y no se notaron diferencias entre quienes comenzaban en el medio laboral y quienes ya tenían experiencia laboral; tampoco entre instituciones públicas ni privadas.

Por su parte, uno de los estudios más amplios a nivel mundial sobre el “éxito” de proyectos de software lo realiza anualmente la empresa Standish Group desde 1994. Para 2015, 29% fueron exitosos, el 19% cancelados y el 52% “discutidos” (Laboratorio de las TI, 2016), de un universo de 50,000 proyectos. Los niveles entre 2011 y 2015 fueron similares.

Aunque no había habido un seguimiento formal en la UPIICSA en torno a errores metodológicos, puede suponerse que se trata de un problema crónico con base en los comentarios surgidos en las juntas de los encargados de seminarios de titulación y los jefes de carrera. Por citar un ejemplo: por lo común hay inconsistencias notables entre la estructura organización de la empresa en que se hace una tesina, los requerimientos del sistema de información a desarrollar, el diseño de pantallas y la estructura interna del software terminado. Dichas inconsistencias no tiene que ver directamente con la codificación, sino con el desarrollo del software en su conjunto. En términos muy generales, la relación de la codificación con las otras disciplinas que la rodean: Ingeniería de Requerimientos, Diseño de Base de Datos, Formulación y Evaluación de Proyectos y Análisis de Sistemas, entre otras.

ISSN: 2448 - 6574

Estamos, pues ante dos vertientes de fallas delicadas en el aprendizaje significativo de la programación. Los relativos a codificación se mencionan comúnmente, pero los relacionados a aspectos metodológicos suelen pasar inadvertidos. Más allá de la evaluación técnica, existen elementos que normalmente no forman parte de ningún examen, pero que indudablemente deben considerarse como parte de la calidad del software.

En este caso, se abordará la situación de los errores metodológicos en el curso de Lógica de Programación: aquellas fallas en el producto software que no dependen directamente de la aplicación de código. Básicamente, si no existe confusión en la interpretación de los requerimientos, la consideración de posibles casos especiales, la presentación correcta de las pantallas al usuario y la validación de posibles datos de entrada inconsistente.

Justificación

El problema es de gran relevancia. Tan solo en México, hubo 227,675 estudiantes de informática a nivel superior en escuelas públicas en el ciclo escolar 2016-2017, una de las disciplinas más demandadas a nivel nacional (ANUIES, Anuario de Educación Superior – Licenciatura, 2017).

Es necesario ahondar sobre el tema de manera pormenorizada y verificable para llegar a vislumbrar las diferentes aristas de este problema crónico, así como posibles estrategias de solución. Los ejercicios aquí expresados se realizaron en las dos carreras de informática de la UPIICSA. Pudiera parecer ser muy limitado, pero no hay que descartar que varias conclusiones puedan tomarse como punto de partida para estudios de nacional e incluso internacional.

Objetivo del proyecto

Estimar la magnitud de errores metodológicos en el desarrollo de los programas al término del curso de Lógica de Programación en las carreras de Informática de la UPIICSA.

ISSN: 2448 - 6574

Marco teórico

Existen pocos acercamientos de lo que debe ser la calidad del software y los criterios metodológicos en un contexto de enseñanza de la programación, sobre todo en los cursos iniciales. Así lo hace ver un recorrido general de las ponencias en congresos sobre computación y educación, los libros de texto de editoriales más reconocidas y las revistas académicas y de divulgación sobre computación e informática. Adicionalmente, representa un punto de intersección entre la programación y la Ingeniería de Software, un "punto ciego" entre dos ramas del conocimiento que no es abordado por libros de programación, de calidad ni de Ingeniería de Software.

Diversos autores han trabajado sobre un intento de definición sencilla del término calidad. Martínez (2015) sugiere que la visión "ingenieril" debiera complementarse con una visión "de usuario". De hecho, abordarse simultáneamente:

> "El testing nos permite validar si un producto cumple o no un requerimiento funcional y otros atributos de calidad definidos para él… Los procesos nos indican cuál es la forma recomendada para realizar una actividad o conjunto de ellas… [Mientras que] las certificaciones nos permiten demostrar el conocimiento y aplicación de mejores prácticas"… Sin embargo [en otra vertiente] para el cliente un producto o servicio se considerará de calidad, si cumple con sus expectativas. Y estará dispuesto a "pagar" por ello, en el sentido más amplio, si esto ocurre.

Martínez (2015) ha asentado varios enfoques. Uno de ellos son los atributos de calidad del producto, incluyendo los conocimientos que se requieren para construirlo guiados por las llamadas "mejores prácticas". Es el llamado enfoque "ingenieril". Desde el punto de vista metodológico, los procesos que deben seguirse para alcanzar este objetivo, así como el cumplimiento de la expectativa del cliente. El diseño y aplicación de las pruebas al producto pertenecen a ambos ámbitos: por una parte, son parte del proceso; por otra, es una garantía del producto.

Suárez (2014, p.23) coincide con este enfoque y clasifica la calidad del software bajo tres enfoques:

> 1.- La calidad del proceso. "La calidad vista desde el mundo de los procesos nos dice que la calidad del producto software está determinada por la calidad del proceso. Por proceso se entienden las actividades, tareas, entrada, salida, procedimientos, etc., para desarrollar y

ISSN: 2448 - 6574

mantener software… Modelos, normas y metodologías típicas aquí son CMMI, ISO 15504 / ISO 12207, el ciclo de vida usado; incluso las metodologías ágiles entran aquí."

2.- La calidad del producto. Existen modelos de calidad de producto, destacando entre ellos la ISO 9126 (ISO, 2001), o la nueva serie ISO 25000 (ISO, 2005a), que especifica diferentes dimensiones de la calidad de producto. Aunque aquí la dura tarea de evaluación recae en el uso de métricas software.

3.- La calidad del equipo de trabajo. "Podemos encontrar decenas de aproximaciones para mejorar la calidad de las personas, que van desde el tan de moda coaching, a la filosofía ágil de lograr la auto-organización de los equipos, estrategias de motivación, combinaciones de los anteriores, etc. E incluso hasta hay modelos, como son los TSP y PSP."

En cuanto a los libros de texto, solo uno de ellos brinda un acercamiento hacia criterios de calidad del software. López y Gutiérrez (2014, p. 229-235) sugieren distinguir entre cualidades que rodean al software (coherencia entre el software y los objetivos de la empresa, costo de compra-venta razonable y garantía de soporte y actualización); cualidades del software como producto (funcionalidad, corrección, robustez, facilidad de uso, portabilidad, eficiencia, seguridad, facilidad de mantenimiento) y cualidades del proceso de construcción (oportunidad y economía). Los mismos autores reconocen los elementos que tomaron de Meyer (1999). El año de edición da idea de uno de los temas más importantes sobre desarrollo de software. En ese sentido, sorprende que en términos generales, los libros de texto no reflejan definiciones de calidad ni metodologías de trabajo explícitas. Que solo un libro de las editoriales más reconocidas lo haya retomado en más de quince años refleja un vacío realmente grave.

Resultados

Las pruebas se basaron sobre la aplicación de problemas en el repaso sobre Lógica de Programación al iniciar el curso de Algoritmos Computacionales, inmediatamente subsecuente al de Lógica de Programación, que inicia el área de programación en las carreras y no tiene ningún conocimiento antecedente directo. Su diseño fue realizado considerando los distintos enfoques de la calidad del software, pero haciendo hincapié en la calidad del producto. Se trató de "medir" en qué grado se cumplía el objetivo final. Por el momento, no se abordaron los aspectos del proceso que siguieron los estudiantes para el desarrollo del programa. En cuanto

ISSN: 2448 - 6574

a los criterios de calidad del producto, se hizo hincapié en la corrección (que se ejecutara correctamente conforme a las especificaciones) y se abordó el problema de las validaciones, por considerar que estas abren un riesgo muy fuerte en el funcionamiento.

Descuento sobre descuento. El primer acercamiento a la problemática desde esta perspectiva se dio en febrero de 2016, cuyos resultados se sintetizan en el cuadro 1. Se expuso por escrito el problema y se dio la oportunidad de resolver cualquier duda sobre el requerimiento. El texto fue:

> Hacer un programa que calcule el precio final de un producto con "rebaja sobre rebaja". El precio original lo dará el usuario, así como ambos descuentos en porcentaje.

Como podrá observarse, 35% no logró hacer el programa, lo que es consistente con algunas evaluaciones diagnósticas de Algoritmos Computacionales en torno a los conocimientos que se aprendieron en Lógica de Programación relativos a codificación.

Pero la vertiente metodológica –que no se había abordado hasta este momento- es igualmente relevante. 55% hizo el programa fuera de las especificaciones señaladas y 75% de quienes lograron hacer el código no validó. **Únicamente 10% de los trabajos podría aceptarse como un software de calidad.** Eso no se debe a la falta de conocimientos sobre codificación.

En resumen, si se tuvieran que valorar los ejercicios de programación después de cursar la unidad de aprendizaje de Lógica de Programación en la UPIICSA conforme a estándares básicos de calidad del producto software, prácticamente el 90% tendrían que ser reprocesados por una o más situaciones significativas que corregir. La problemática, delicada en cualquier asignatura, lo es aún más en la primera que aborda la programación y que debiera sentar las bases de las buenas prácticas en el desarrollo de software.

Con ese antecedente, durante agosto y septiembre de 2017 se trató de estimar el porcentaje de programas que se tendrían que volver a realizar debido justamente, a fallas metodológicas. En las dos evaluaciones de este tipo que se realizaron más del 88% de los ejercicios presentaron al menos una falla delicada en el programa. En otras palabras, cuando menos 9 de cada 10 "productos finales" tendrían que ser reprocesados.

ISSN: 2448 - 6574

¿El programa se ejecutó?	Conforme a requerimientos	Fuera de requerimientos	Total
Sí	Con validaciones 3 Sin validaciones 7	Con validaciones 2 Sin validaciones 8	20
No	4	7	11
Total	14	17	31

Cuadro 1. Primer acercamiento a un diagnóstico sobre problemas metodológicos sobre temas de Lógica de Programación aplicado a estudiantes de Algoritmos Computaciones en 2016

Un tinaco cilíndrico. Se hizo un primer acercamiento en agosto de 2017, en una secuencia de Algoritmos Computacionales de la carrera de la Ing. Informática, con el siguiente problema:

> Realizar un programa que reciba el diámetro y altura en centímetros de un tinaco cilíndrico y despliegue su capacidad en litros.

El trabajo se presentaría cuando a juicio de los alumnos ya estuviera terminado el programa y listo para presentarlo al usuario, incluyendo una pantalla con un ejemplo representativo real. De un total de 16 trabajos, 11 tenían un error de cálculo (69%); 6 cambiaron especificaciones, sobre todo en la unidad de medida (38%); y 4 pusieron datos no representativos (38%), lo cual no afecta al cálculo pero dificultará en un futuro las explicaciones hacia el usuario final. Únicamente 3 resultaron impecables (19%). La prueba con la secuencia de la Lic. en Ciencias de la Informática arrojó valores muy similares (ver figura 1).

ISSN: 2448 - 6574

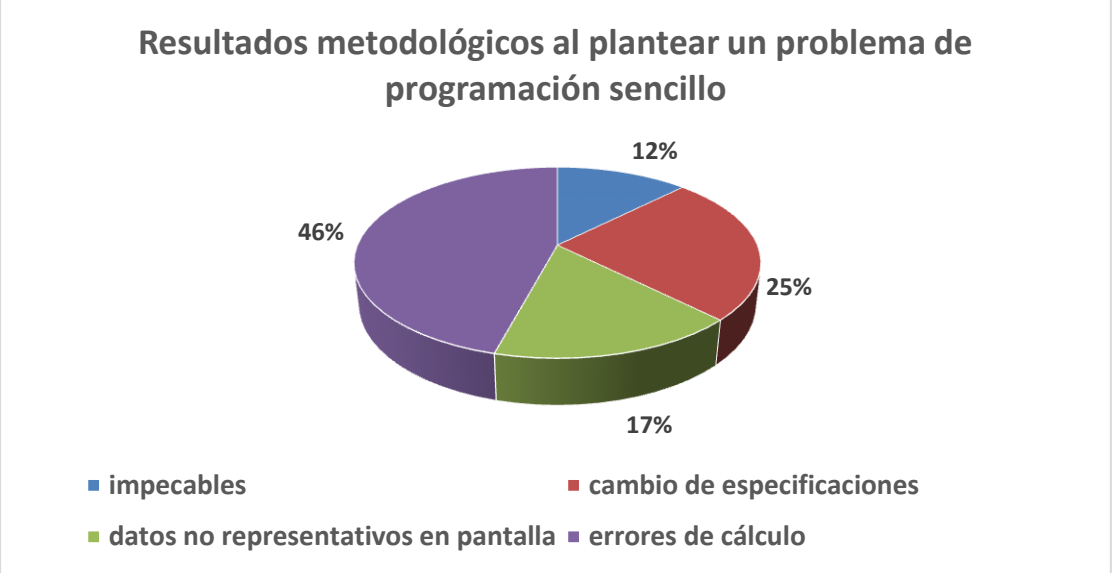

Figura 1. Situación de aspectos metodológicos en septiembre de 2017

en el caso de la capacidad de un tinaco cilíndrico.

Cálculo del factorial. Se planteó otro problema aplicado a la Lic. en Ciencias de la Informática: el cálculo del factorial. En este caso, se evaluó en cuantos intentos el programa se ejecutaba adecuadamente en diecinueve programas. dos lo hicieron en un intento; uno en dos intentos; tres en tres intentos; cuatro en cuatro intentos. En promedio a la segunda corrección ante el “usuario". Únicamente uno de los once consideró el caso especial del factorial de cero, cuyo resultado es uno. Ninguno validó que no se introdujeran negativos. Después de dos correcciones ante el “usuario”, únicamente 45% trabajarían bien en el caso normal y 9% en el caso excepcional del factorial de cero. Ninguno tendría validación.

En la otra carrera, Ing. en Informática, se les pidió hacer el programa “contra-reloj”. Los diez primeros lo hicieron en un tiempo excepcional de 6 minutos 28 segundos, aunque únicamente dos previeron el caso especial del factorial de cero y ninguno validó. Por el tiempo obtenido se trata de alumnos que tienen un nivel de codificación muy alto, pero sin la costumbre de vislumbrar los casos especiales ni validar. Sí se nota una diferencia entre los alumnos de ambas carreras en los aspectos de codificación (Ing. en Informática tiene mayor nivel), pero no en los aspectos metodológicos.

ISSN: 2448 - 6574

La mayor de tres calificaciones. El primer ejercicio fue elaborar un programa que recibiera del usuario tres calificaciones departamentales (parciales) y a partir de ellas obtuviera el promedio final a un decimal, indicara si el alumno aprobó y, finalmente, dijera cual fue la mayor. El objetivo era tener un programa que trabajara conforme a requerimientos; en cada revisión solo se señalaría un error. El límite de tiempo fue de una hora, más que suficiente para el nivel que deben tener. De los que terminaron en tiempo, únicamente 43% lo hicieron bien. El principal motivo fue que no pensaron en que las calificaciones pudieran ser iguales; en otros casos no se expresó el promedio a un decimal o se omitió; también sucedió que la condición de aprobación se puso con las instrucción `promedio > 5`, lo cual arrojaría a un 5.3 como aprobado (la instrucción correcta era `promedio >= 6)`.

Los errores en sí son “normales” y muy comunes de cometer. Lo interesante es ¿por qué no se detectaron si se tuvo tiempo de sobra para revisar? Aún más, ¿por qué no se sometió a una revisión con otro compañero si se tenía esa oportunidad? La inclusión de un “tester”, equivalente de alguna forma a una revisión técnica de pares, no suele darse en el área de programación a pesar que suele ser una recomendación básica para el desarrollo de software.

Conclusiones

Los resultados sintetizados de las pruebas fueron los siguientes:

- “Descuento sobre descuento” 35% no tuvieron los conocimientos sobre codificación para realizarlo, 55% lo hizo fuera de requerimientos y 75% no puso validaciones.
- Tinaco. 69% con error de cálculo; 38% sin datos representativos (se ejecutaría correctamente pero la pantalla sería poco intuitiva hacia el usuario).
- Factorial sin límite de tiempo: únicamente 45% trabajaban bien al escribirse en papel después de tres revisiones y al menos el señalamiento de dos errores específicos.
- Evaluación de factorial en codificación “contra-reloj”. Se tomaron los mejores diez trabajos: todos corrieron adecuadamente en el caso normal, pero únicamente dos consideraron el caso especial de factorial de cero y ninguno validó.
- Mayor de tres calificaciones, 57% no consideraron correctamente el caso de los iguales, omitieron especificaciones o el formato esperado.

ISSN: 2448 - 6574

Todos los ejercicios apuntan hacia la falta de un apego cuidadoso de los requerimientos, la ausencia de una revisión más allá del propio programador, no cuidar casos especiales y no validar los datos de entrada. No se nota una diferencia significativa en estos aspectos entre quienes saben codificar y quienes llevan un curso de programación por primera vez.

En ese sentido, se debe insistir en una última conclusión: una situación tan generalizada no puede atribuirse al estudiante. Los errores metodológicos son mayormente un problema de enseñanza. Es indispensable que los maestros dejemos de decir que los alumnos "no saben programar" para referirnos a un problema mucho mayor: aún no sabemos cómo enseñar el desarrollo de software. Por las estimaciones internacionales, parece ser un problema de muy alta envergadura, que pareciera, hasta hoy como docentes hemos eludido y minimizado.

Referencias bibliográficas

Asociación Nacional de Universidades e Instituciones de Educación Superior, ANUIES (2017). *Anuario Estadístico de Educación Superior*. México: ANUIES [recuperado el 28 de enero de 2018 de http://www.anuies.mx/informacion-y-servicios/informacion-estadistica-de-educacion-superior/anuario-estadistico-de-educacion-superior].

Laboratorio de las TI (2016). *La Teoría del Caos 2015*. [Recuperado de http://www.laboratorioti.com/2016/05/16/informe-del-caos-2015-chaos-report-2015-bien-mal-fueron-los-proyectos-ano-2015 el día 16 de julio de 2017].

López, J.; Gutiérrez, G. (2014). *Programación Orientada a Objetos con C++ y Java*. México: Patria.

Martínez, R. (2015). Calidad no es testing, ni procesos ni certificaciones. *Software Gurú #48*. México. [recuperado de https://sg.com.mx/revista/48/calidad-no-es-testing-ni-procesos-ni-certificaciones#.WW5LB4jhDIV el día 15 de julio de 2017].

Suárez (2014). I Jornada sobre Calidad del Producto Software e ISO 25000, Santiago de Compostela, 10 de junio de 2014. España: Colexio Profesional de Enxeñaría en Informática de Galica. [recuperado de https://www.cpeig.gal/portal/system/files/Libro+Jornadas+Galicia+Calidad+Software.pdf el día 22 de abril de 2018].