Actas de las Jenui, vol. 4. 2019.
Páginas: 127-134

Evaluación de las habilidades en programación de los estudiantes del grado en Ingeniería Informática de la Universidad de Alicante: un estudio empírico

Juan Antonio Pérez-Ortiz[1], David Tomás[1], Cristina Cachero[1], Domingo Gallardo[2]

[1] Dept. Llenguatges i Sistemes Informàtics, Universitat d'Alacant, España
[2] Dept. Ciències de la Computació i Intel·ligència Artificial, Universitat d'Alacant, España

{japerez,dtomas,ccachero}@dlsi.ua.es, domingo@dccia.ua.es

Resumen

En este trabajo analizamos las habilidades como programadores de los alumnos de cuarto curso del grado en Ingeniería Informática de la Universidad de Alicante con el propósito de identificar carencias comunes en su formación que no hayan sido abordadas adecuadamente en el transcurso de la titulación. Para tal fin, presentamos una prueba de evaluación de habilidades en programación y mostramos los resultados obtenidos en ella por 88 estudiantes. Un análisis de los resultados por itinerario muestra diferencias significativas dependiendo de la especialidad cursada.

Abstract

In this work the abilities as programmers of the fourth-year students (almost graduates) of the Computer Engineering degree at University of Alicante is analysed, with the aim of identifying deficiencies in their training. A test to evaluate the programming skills of the students is introduced in this paper, and the results obtained by 88 students are presented. An analysis of the results by specialisation stream shows significant differences depending on the specialisation taken by the students.

Palabras clave

Evaluación de habilidades en programación, pensamiento computacional, estudio empírico en docencia universitaria.

1. Introducción

La programación es una de las tareas a la que más frecuentemente se ha de dedicar un ingeniero informático durante su carrera profesional, tal y como se desprende, entre otros, del informe [4] que sirve de guía para la elaboración de muchos planes de estudio de informática en todo el mundo. Este informe considera la programación como el área de conocimiento que merece más atención en los cinco itinerarios definidos para la titulación: Ingeniería de Computadores (IC), Tecnologías de la Información (TI), Computación (C), Sistemas de Información (SI) e Ingeniería del Software (IS). Es fundamental, por ello, que los titulados adquieran una formación sólida en un área transversal a muchas asignaturas del grado.

Sin embargo, a pesar de esta importancia reconocida, la percepción de algunos profesores de último curso del grado en Ingeniería Informática (GII) de la Universidad de Alicante es que parte de su alumnado tiene dificultades con habilidades que podrían considerarse esenciales para un buen programador, como, entre otras, extender un código propio o ajeno, relacionar conceptos entre diferentes lenguajes de programación, entender lo que el compilador y el entorno de ejecución hacen con su programa, o depurar código de manera adecuada.

Para contrastar científicamente esta percepción subjetiva, en 2017 se creó una red docente con 11 profesores del GII de la Universidad de Alicante que impartían docencia en asignaturas de programación. El propósito de la red fue definir y ejecutar un procedimiento para determinar objetivamente si los estudiantes de último curso adquieren las habilidades en programación esperables tras su paso por la universidad. Un segundo objetivo fue determinar si estas habilidades eran equivalentes en los distintos itinerarios de la titulación. El procedimiento se planteó como una evaluación transversal y no de los conocimientos de una asignatura concreta.

Este artículo presenta los principales resultados del trabajo de dicha red. En la próxima sección se introducen los trabajos relacionados; los principios de diseño del instrumento de evaluación y su contenido pueden verse en la sección 3 y en el apéndice A, respectivamente; la sección 4 describe la ejecución del estudio; la sección 5 muestra el análisis descriptivo, los índices de dificultad y la validez discriminante de los distintos ítems de la prueba; la sección 6 presenta el análisis de

los datos por itinerario; finalmente, la sección 7 recoge las principales conclusiones y trabajos futuros.

2. Trabajo relacionado

La evaluación de las habilidades en programación constituye un tema abierto para el que la comunidad educativa todavía no posee unos recursos estándar [3]; se ha llegado incluso a plantear que las formas habituales de evaluación (exámenes y prácticas de programación) no son las más adecuadas [2]. La organización privada ETS (responsable de la conocida prueba de inglés TOEFL) dispone del Major Field Test for Computer Science,[1] un sistema de evaluación de pago que permite comparar los resultados con los obtenidos por otras instituciones. Existen numerosas propuestas de colecciones de problemas de programación más o menos complejos, algunas de las cuales forman parte de competiciones (como la ACM International Collegiate Programming Contest,[2] el Google Code Jam[3] o CodeChef[4]), de procesos de selección de personal (como CodeVue[5] o CoderByte[6]) o de iniciativas docentes (como el proyecto Quantum[7] o el Advanced Placement Program[8]). No existen, no obstante, instrumentos de evaluación estandarizados para estas habilidades.

3. Diseño del instrumento

La carencia de instrumentos estandarizados de medición de habilidades genéricas en programación en el ámbito universitario planteó la necesidad de elaborar uno propio que se ajustara a los objetivos de investigación del presente trabajo. Para ello, se analizaron diferentes aspectos como, entre otros, la manera en la que realizar la evaluación (pruebas individuales o por equipos; concursos de programación; entrevistas; sesiones prácticas frente al ordenador; grabación en vídeo de sesiones de escritura de código; etc.) o el tipo de instrumento (escribir código; ordenar líneas de código [8]; rellenar huecos en programas incompletos; plantear algoritmos que resuelvan problemas en el menor tiempo posible; hacer trazas de programas ya escritos; pruebas de respuesta cerrada o abierta; etc.).

Como resultado del análisis anterior, se diseñó un *test de habilidades de programación* (THP), que puede ser consultado en el apéndice A. Se trata de una única prueba de 50 minutos[9] pensada para estudiantes de cuarto curso del GII. Está formada por 11 preguntas de respuesta abierta pero idealmente única. De esta manera se reduce la probabilidad de acierto por azar, se obliga al estudiante a encontrar la solución por sí mismo sin tener que elegir de entre un conjunto reducido de opciones y se simplifica la corrección de la prueba. Se diseñaron preguntas que cubren diversas *microcompetencias* relacionadas con la programación estructurada y la programación orientada a objetos. Estas microcompetencias y las preguntas de la prueba que las evalúan son las siguientes: hacer trazas y depurar errores de algoritmos iterativos y recursivos (1,2,3,5,7,8); entender cómo avanza y cuándo acaba un bucle (1,3,7,8,11); entender el funcionamiento de las instrucciones condicionales y su anidamiento (2,3,5,6,7,8,11); determinar el valor de expresiones booleanas con operadores lógicos (3); identificar el caso base y el general de una solución recursiva (5,6); determinar la complejidad temporal de un algoritmo (9); entender cómo iterar sobre los elementos de una lista en base a un determinado objetivo (3,5,6,7,11); saber interpretar la especificación de una determinada característica de un lenguaje de programación (4); discernir entre soluciones algorítmicas correctas e incorrectas a un problema dado (5,6,7,11); diferenciar entre clase base y derivada en relaciones de herencia (8); determinar cuándo se utiliza enlace estático o dinámico en contextos de variables polimórficas (8); entender una especificación de un problema y encontrar la solución (9,10); determinar el valor devuelto por la llamada a una función (5,6,7,8,11); manipular estructuras de datos básicas (4,5,6,7,9,11).

Las preguntas no versan sobre ningún lenguaje de programación concreto (se planteó un lenguaje ad hoc para la prueba, mezcla de otros del estilo de Python o Java), ya que no existe ninguno para el que se pueda asegurar un grado de destreza elevado independientemente del itinerario cursado y de la experiencia de cada estudiante. Antes de liberar el instrumento definitivo, se pasó un prototipo de la prueba a una programadora profesional y a tres alumnos de cuarto curso, lo que permitió detectar ambigüedades o inexactitudes, así como ajustar los tiempos y las preguntas.

4. Ejecución de la prueba

El THP se realizó en papel (para realizarlo indistintamente en clase de prácticas o teoría, donde los alumnos no siempre tienen acceso a un ordenador, y para que no pudieran valerse del ordenador para contestar a algunas preguntas) en sesiones de clase durante la tercera semana de mayo de 2018. La prueba se contestó de manera anónima. Durante los 50 minutos de duración de la prueba no se permitió hacer preguntas al profesor

[1] www.ets.org/mft/about/content
[2] icpc.baylor.edu
[3] code.google.com/codejam/
[4] www.codechef.com
[5] www.hirevue.com/products/assessments
[6] coderbyte.com
[7] diagnosticquestions.com/Quantum
[8] apcentral.collegeboard.org/courses
[9] Aunque el tiempo disponible es ajustado, lo consideramos como una variable más del estudio, ya que hubo estudiantes que tuvieron tiempo de contestar a todas las preguntas.

para no sesgar los resultados de un grupo a otro.

Dos instructores supervisaron cada sesión con el fin de evitar cualquier tipo de interacción entre los sujetos. Un total de 88 alumnos realizaron la prueba. Por razones éticas, al inicio de la sesión se solicitó de manera explícita el permiso de los participantes para tratar sus datos de manera anónima y agregada. Todos los estudiantes aceptaron participar. La distribución de los participantes por itinerario puede verse en el cuadro 1.

Itinerario	IC	TI	C	SI	IS
Evaluados	6	25	23	5	29

Cuadro 1: Distribución de los estudiantes evaluados por itinerarios: ingeniería de computadores (IC), tecnologías de la información (TI), computación (C), sistemas de información (SI), ingeniería del software (IS).

La corrección del THP se realizó de forma manual. Con el fin de mitigar el efecto que los distintos correctores pudieran tener en los resultados, se usó una plantilla de corrección con las posibles respuestas correctas. Para elaborarla, se estudiaron las respuestas una a una para dar por buenas en casos muy puntuales soluciones *alternativas*, aunque la prueba se había diseñado intentando que cada respuesta correcta fuera única.

5. Análisis de los resultados

La mediana del número de aciertos fue 5 con una desviación típica (DT) de 2. La mediana del número de respuestas en blanco fue 2 y la del número de respuestas incorrectas fue 4. Todos los estudiantes acertaron como mínimo 2 preguntas, y ninguno acertó más de 9 de las 11 existentes.

El cuadro 2 muestra la cantidad de aciertos, fallos y respuestas en blanco para cada pregunta de la prueba. Las preguntas se ordenaron por dificultad creciente en base al criterio de los autores. Los porcentajes de acierto parecen corroborar esta idea, pero debe tenerse en cuenta que los resultados obtenidos pueden también deberse a que los estudiantes no tenían tiempo para hacer toda la prueba y dejaban las preguntas del final sin abordar.[10] Se puede observar, en cualquier caso, que hay un salto cuantitativo en la cantidad de respuestas en blanco correspondiente a la última pregunta en comparación con las anteriores, lo que nos hace suponer que la mayoría de estudiantes tuvo tiempo de explorar, como mínimo, las preguntas 1 a 10.

El cuadro 2 muestra también el valor del índice de dificultad (Df) y de dos índices de discriminación (Dc_1 y Dc_2) para cada pregunta [7]. Todos estos índices se han calculado sobre la submuestra correspondiente al 27 % de los mejores resultados y al 27 % de los peores.[11] El número de sujetos en cada grupo es de 24.

El índice de dificultad Df indica la proporción de aciertos en la submuestra. Un valor de Df mayor indica una pregunta más fácil. Las tres primeras preguntas del test son intencionadamente muy fáciles (son preguntas sobre variables, bucles y condicionales que incluso un alumno de primer curso debería poder contestar correctamente sin problemas). Los valores en torno a 0,8 de estas tres primeras preguntas son un indicador de su baja dificultad, pero aun así aproximadamente un 20 % de los estudiantes de la submuestra la fallaron (prácticamente todos pertenecían al subgrupo inferior).

El índice de discriminación Dc_1 es la diferencia entre la proporción de aciertos del grupo superior y del inferior. Valores cercanos a 1 indican que hay muchas más respuestas correctas en el grupo superior que en el inferior y que, por tanto, la pregunta contribuye a discriminar entre los alumnos de ambos grupos. Las preguntas con valores de Dc_1 negativos favorecen al grupo inferior y suelen estar mal formuladas; no hay ninguna pregunta de este tipo en nuestra prueba. Del estudio de las fórmulas para el cálculo de Df y Dc_1 se deduce que las preguntas que discriminan mucho (valores de Dc_1 cercanos a 1) no implican gran dificultad (las han contestado bien la mayoría de los alumnos del grupo superior). Esto es una limitación del índice Dc_1, ya que su poder de discriminación se reduce para el caso de preguntas difíciles como las preguntas 8 a 11 de la prueba, que tienen valores relativamente bajos de Dc_1 si se comparan con los valores relativamente altos de las preguntas 4 a 7, que son muy discriminantes.

Para evitar el problema de que Dc_1 solo puede llegar a valer 1 si ningún estudiante del grupo superior falla la pregunta (o, en general, que solo toma valores cercanos a 1 cuando las preguntas no conllevan dificultad), se muestra también en el cuadro 2 un segundo índice de discriminación Dc_2 que es independiente del grado de dificultad de la pregunta. Este índice se calcula como la proporción de aciertos en el grupo superior con respecto al número total de acertantes y vale 1 si todos los acertantes (independientemente de su número) pertenecen al grupo superior. Como se ve, Dc_2 toma valores muy altos para todas las preguntas desde la 4, llegando incluso al máximo valor de 1 para las preguntas 7, 9, 10 y 11, que no fueron acertadas por una parte importante de los estudiantes del grupo superior, aunque, al mismo tiempo, no hubo ningún estudiante del grupo inferior que las contestara correctamente. Dc_2 es muy discriminatorio incluso para las tres primeras preguntas.

Las figuras 1 y 2 muestran, respectivamente, la distribución del número de aciertos y fallos en cada itinerario. Téngase en cuenta que los itinerarios IC y SI tienen

[10] Se sugería a los estudiantes que resolvieran las preguntas en el orden en que estas aparecían en la prueba.

[11] A la hora de ordenar, en caso de empate en el número de aciertos, se consideraba mejor resultado el que tenía menos fallos.

Pregunta	1	2	3	4	5	6	7	8	9	10	11
Bien	71	78	75	56	45	52	25	12	10	8	6
Blanco	1	0	0	0	8	8	21	30	36	27	59
Mal	17	11	14	33	36	29	43	47	43	54	24
Acierto (%)	79,8	87,6	84,3	62,9	50,6	58,4	28,1	13,5	11,2	9,0	6,7
Df	0,81	0,81	0,77	0,65	0,48	0,58	0,33	0,19	0,13	0,10	0,08
Dc_1	0,25	0,25	0,42	0,63	0,88	0,67	0,63	0,25	0,21	0,21	0,17
Dc_2	0,56	0,56	0,62	0,74	0,96	0,79	1,00	0,89	1,00	1,00	1,00

Cuadro 2: Resultados de la prueba por pregunta con índices de dificultad y discriminación

muy pocos alumnos matriculados, lo que puede afectar a la generalización de los resultados. En las figuras 1 y 2 la línea horizontal inferior de cada caja representa el cuartil Q1, la línea superior el cuartil Q3 y la línea del interior el cuartil Q2 o mediana. Los brazos señalan los valores mínimo y máximo del conjunto de respuestas. No hay valores atípicos en ninguno de los itinerarios, definidos como valores que se alejan más de 1.5 veces del rango intercuartil de la muestra.

6. Análisis por itinerario

Una vez calculados los estadísticos descriptivos relevantes y comprobado el índice de dificultad y la validez discriminante de las preguntas, se procedió a analizar los datos para averiguar si las capacidades de los alumnos variaban de manera significativa en función del itinerario escogido y, si esto era así, qué itinerarios eran los que causaban las diferencias observadas.

Variables y escalas de medición. Dado el bajo número de sujetos de los itinerarios de IC (6) y SI (5), y las grandes diferencias con respecto al resto de itinerarios, se decidió dejarlos fuera del estudio estadístico. Se definió por tanto una sola variable independiente (VI), *itinerario*, de tipo categorial inter-sujeto, con tres posibles valores: IS, C y TI. Como variables dependientes (VD) se definieron las siguientes variables y medidas de eficacia:

- NRC: número de respuestas correctas: medida de tipo ratio. Instrumento de medición: escala de 11 ítems con rango $[0, 11]$
- NRI: número de respuestas incorrectas: medida de tipo ratio. Instrumento de medición: escala de 11 ítems con rango $[0, 11]$

Hipótesis. A partir de las variables y medidas definidas, se establecieron las siguientes hipótesis nulas, susceptibles de ser refutadas estadísticamente mediante el método de refutación de hipótesis:

- HItinerarioNRC$_0$: la puntuación media de NRC es la misma independientemente del itinerario ($\mu_1 = \mu_2 = \mu_3$). La correspondiente hipótesis alternativa establece que al menos una de las medias es distinta.
- HItinerarioNRI$_0$: la puntuación media de NRI es la misma independientemente del itinerario ($\mu_1 = \mu_2 = \mu_3$). Al igual que en el caso anterior, la hipótesis alternativa es que al menos una de las puntuaciones medias es distinta.

Análisis estadístico. Para el análisis de los datos se utilizó el software SPSS Statistics v.23. El cuadro 3 muestra la media y DT de las medidas del estudio.

	NRC		NRI	
	Media	DT	Media	DT
IS	5,38	1,92	3,28	1,94
C	5,70	1,94	3,35	1,90
TI	4,16	1,84	4,88	2,19

Cuadro 3: Estadísticos descriptivos por itinerario

Con el fin de analizar las hipótesis HItinerarioNRC$_0$ y HItinerarioNRI$_0$ se aplicaron sendos *one-way* ANOVA, seguidos de comparaciones por pares mediante el test *post-hoc* de Tukey-Kramer (apropiado cuando el diseño no está balanceado) para ver qué itinerarios son los que causan las diferencias significativas. El *one-way* ANOVA tiene seis asunciones que deben ser revisadas. Las primeras tres asunciones están relacionadas con el diseño del estudio: se requiere una variable dependiente medida a nivel continuo (en este caso NRC y NRI), una variable independiente de tipo categorial con tres o más grupos independientes (en este caso el itinerario) e independencia de observaciones, i. e., que no haya relación entre las observaciones en cada grupo de la variable independiente o entre grupos. Las otras tres asunciones se refieren a cómo los datos se adaptan a las características del modelo *one-way* ANOVA para permitir la producción de un resultado válido. Éstas son:

- Ausencia de valores extremos en los dos grupos de la variable independiente en términos de la variable dependiente. Un examen visual de los diagramas de caja para NRC y NRI (véase las figuras 1 y 2) muestra cómo no existen valores situados más

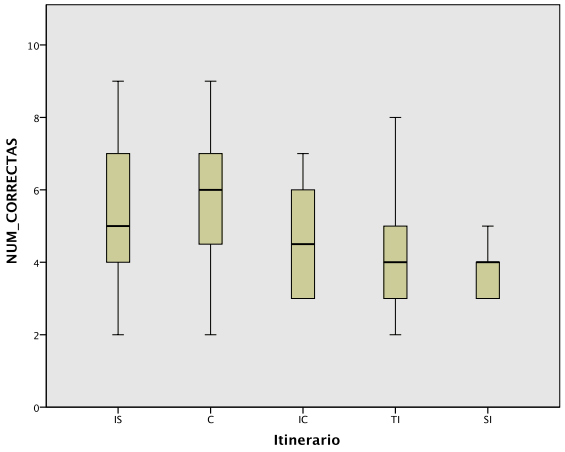

Figura 1: NRC: diagrama de caja

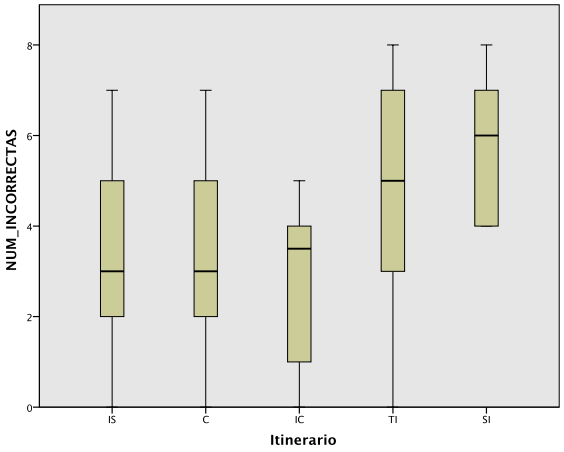

Figura 2: NRI: diagrama de caja

allá de 1,5 veces la longitud de la caja medido a partir de su borde para ninguno de los grupos.

- Distribución aproximadamente normal para cada grupo de la VD. Esta asunción se cumple para las variable NRI, como demuestra el resultado del test de Shapiro-Wilk ($\rho > 0{,}05$ para las tres celdas del diseño). Sin embargo, para la variable NRC la asunción se viola para el grupo TI ($\rho = 0{,}023$).
- Homogeneidad de varianzas. Esta asunción se ha comprobado mediante el test de Levene. En este caso, el test ha arrojado valores de $\rho > 0{,}05$ para todas las celdas de la variable NRI, pero no para las celdas de la variable NRC ($\rho = 0{,}041$).

Dadas las violaciones que se producen con la variable NRC, y dado el ligero sesgo positivo de las distribuciones del itinerario TI (el que más problemas presenta), se aplicó sobre ella una transformación de raíz cuadrada. Con esta transformación, la nueva variable SQRT_NRC cumple la asunción de homogeneidad de varianzas ($\rho > 0{,}05$ para todas las celdas), aunque su distribución sigue sin ser normal según el test de Shapiro-Wilk ($\rho = 0{,}034$). No obstante, el *one-way* ANOVA es robusto con respecto a la violación de normalidad, sobre todo cuando los tamaños de muestra son similares [6], como es el caso en este estudio. A continuación se describen los resultados con el formato media±DT.[12]

El número medio de respuestas correctas (NRC) más bajo fue el correspondiente al itinerario TI ($n = 25$; $4{,}16 \pm 1{,}84$) seguido por el itinerario IS ($n = 29$; $5{,}38 \pm 1{,}92$) y, con el mejor rendimiento, el itinerario C ($n = 23$; $5{,}70 \pm 1{,}94$). Estas medias son estadísticamente distintas para los distintos itinerarios, $F(2{,}74) = 4{,}735$, $\rho = 0{,}012$, $\eta_p^2 = 0{,}113$. El análisis de los resultados del test *post-hoc* de Tukey-Kramer para la medida SQRT_NRC indica que las diferencias significativas se producen entre los pares TI–IS, con una diferencia de $1{,}604$ (95 % CI; $0{,}29$ a $2{,}92$), y TI–C, con una diferencia de $1{,}532$ (95 % CI; $0{,}14$ a $2{,}92$).

Por lo que respecta al número medio de respuestas incorrectas (NRI), el más alto se produjo nuevamente en el itinerario TI ($n = 25$; $4{,}88 \pm 2{,}19$), seguido por el itinerario C ($n = 23$; $3{,}35 \pm 1{,}90$) y, por último, el itinerario IS ($n = 29$; $3{,}28 \pm 1{,}94$). Estas medias también son estadísticamente distintas para los distintos itinerarios, $F(2{,}74) = 5{,}162$, $\rho = 0{,}008$, $\eta_p^2 = 0{,}122$. El análisis de los resultados del test *post-hoc* de Tukey-Kramer para la medida NRI indica que las diferencias significativas también se producen en los pares TI–IS, con una diferencia de $-0{,}29$ (95 % CI; $-0{,}57$ a $-0{,}006$), y TI–C, con una diferencia de $-0{,}3583$ (95 % CI; $-0{,}66$ a $-0{,}06$).

Con estos resultados se refutan las dos hipótesis nulas del estudio (HItinerarioNRC$_0$ y HItinerarioNRI$_0$) y se aceptan las hipótesis alternativas, i. e., que existen diferencias estadísticamente significativas en cuanto a NRC y NRI entre los grupos integrados por estudiantes de los distintos itinerarios.

Amenazas a la validez. Por último, para el análisis de las principales amenazas a la validez del estudio se ha seguido la clasificación propuesta por Cook y Campbell [1]: amenazas a la validez interna, externa, de constructo y de conclusión. Como principales amenazas a la validez interna hemos detectado el sesgo de selección: la variable itinerario surge de la mera observación, sin aleatorización, por lo que es imposible establecer relaciones de causalidad entre itinerarios y habilidades de programación, sino solo constatar diferencias estadísticamente significativas. Además, todos los sujetos que se presentaron a la sesión del día en el que el estudio estaba programado consintieron participar. El THP no es

[12] Para facilitar su interpretación, cuando se hable de medias y DTs de las variables se reportarán los valores de NRC, aunque para los análisis estadísticos se haya utilizado la variable SQRT_NRC.

un instrumento estandarizado, aunque los análisis realizados muestran que sus ítems tienen una buena validez discriminante. Dado que la prueba incluyó a todos los estudiantes de todos los itinerarios del GII, podemos asumir que no existen amenazas a la validez externa en el contexto de la Universidad de Alicante, aunque sería necesario replicar el estudio en otras universidades. El control de las asunciones de todos los tests estadísticos aplicados mitiga las amenazas a la validez estadística, mientras que la definición clara de hipótesis, variables y medidas mitiga las amenazas a la validez de constructo.

7. Conclusiones y trabajo futuro

Este trabajo constituye una primera aproximación a la evaluación de las destrezas de los estudiantes de último cursos del GII. El desempeño general en alguna de las pruebas ha sido limitado, lo que da soporte a la percepción subjetiva de parte del profesorado de las carencias que sufren algunos alumnos en sus habilidades de programación, en línea con trabajos anteriores [5] que sugieren que la falta de formación en conceptos base de bajo nivel puede ser la razón de estos resultados. Estas deficiencias han sido más acentuadas en el caso del itinerario de TI, resultado que, sin ser esperable, se puede considerar como razonable: la especialidad de TI tiene una carga de programación inferior al resto de itinerarios, con lo que la formación en esta área es menos completa y se atrae también a un perfil de alumnado que prefiere invertir menos tiempo en programar, ya sea por falta de afinidad o por carencias en la materia.

Para avanzar en esta línea de investigación, es necesario seguir trabajando en este tipo de pruebas en tanto no se disponga de un instrumento estandarizado y validado para medir las habilidades de programación en el contexto universitario. En este sentido, pensamos que se hace necesaria la definición de unas *microcompetencias* que guíen de forma más precisa que la señalada en los planes de estudio cuáles son los conocimientos y habilidades que se espera que adquieran los estudiantes. La preparación de una batería mayor de preguntas y la vinculación de estas con las diferentes microcompetencias en programación es indispensable para que la prueba pueda aplicarse a muestras de población mayores y en diferentes años, incluso como parte de estudios longitudinales que documenten la mejora de estas competencias según el alumno avanza de curso. El fin último de los resultados es poder aplicar medidas correctivas a lo largo de la titulación para que los alumnos alcancen las competencias de programación esperadas.

Agradecimientos. Este trabajo ha sido posible gracias a la red docente 4099 subvencionada por la edición 2017–2018 del *Programa I3CE de Investigación en Docencia Universitaria* del Instituto de Ciencias de la Educación de la Universitat d'Alacant. Los autores agradecen su participación en la elaboración de la prueba evaluadora a los otros miembros de dicha red docente (Jorge Calera, Antonio Miguel Corbí, Patricia Compañ, Francisco José Gallego, Faraón Llorens, Francisco Moreno, Jesús Peral y Rosana Satorre), así como a los profesores que cedieron tiempo de sus asignaturas para la realización de la prueba (Patricio Martínez, Higinio Mora, Francisco Moreno y Rafael Muñoz).

Referencias

[1] Thomas D Cook, Donald Thomas Campbell, y Arles Day. *Quasi-experimentation: Design & analysis issues for field settings*, 351. Houghton Mifflin Boston, 1979.

[2] Charlie Daly y John Waldron. Assessing the assessment of programming ability. En *Proc. of the 35th SIGCSE Technical Symposium on Computer Science Education*, 2004.

[3] Peter J. Denning. Remaining trouble spots with computational thinking. *Commun. ACM*, 60, 2017.

[4] The Joint Task Force for Computing Curricula 2005. *Computing Curricula 2005*. 2006.

[5] Francisco J. Gallego, Rosana Satorre, Patricia Compañ, y Carlos J. Villagrá. Explicando el bajo nivel de programación de los estudiantes. *ReVisión*, 11(1), 2018.

[6] Lisa M Lix, Joanne C Keselman, y HJ Keselman. Consequences of assumption violations revisited. *Review of educational research*, 66(4), 1996.

[7] Pedro Morales. Análisis de ítems en las pruebas objetivas. Univ. Pontificia Comillas, Madrid, 2012.

[8] Dale Parsons y Patricia Haden. Parson's programming puzzles: A fun and effective learning tool for first programming courses. En *Proc. of the 8th Australasian Conf. on Computing Education*, 2006.

Apéndice A: test de habilidades en programación (THP)

Además de las 11 preguntas mostradas a continuación, la prueba constaba de una introducción con las instrucciones para su realización y la descripción de las características básicas del lenguaje de programación $\mathcal{L}$, inventado expresamente para la prueba como una mezcla de lenguajes como Python, Java o C. Dicha introducción se ha omitido aquí por falta de espacio.

Pregunta 1. Dado el siguiente código en $\mathcal{L}$, que usa un bucle while convencional, indica cuántas veces se imprimirá la cadena `"Hola"`.

```
i = 1
while i < 22:
```

```
  i = i + 3
  print("Hola")
```

Pregunta 2. Indica cuál sería el valor de `j` tras ejecutar este código escrito en $\mathcal{L}$.

```
j = 4
if j < 5:
  j = j + 6
if j < 10:
  j = j + 4
else:
  j = j + 7
```

Pregunta 3. Tienes que escribir un programa en $\mathcal{L}$ que itere sobre un array `v` que almacena 20 elementos de tipo entero. El objetivo es localizar la primera aparición del valor 100 en el array y, en ese momento, abandonar la búsqueda. El programa nunca puede leer fuera de los límites del array.

¿Cuál de los siguientes bloques de código realiza correctamente la tarea anterior?

```
a) found = False
   i = 0
   while i < 20 and not found:
     if v[i] == 100:
       found = True
     else:
       i = i + 1
b) found = False
   i = 0
   while not found:
     if v[i] == 100:
       found = True
     else:
       i = i + 1
c) found = False
   i = 0
   while i < 20 or not found:
     if v[i] == 100:
       found = True
     else:
       i = i + 1
d) found = False
   i = 0
   while i < 20 or found:
     if v[i] == 100:
       found = True
     else:
       i = i + 1
```

Pregunta 4. El lenguaje de programación $\mathcal{L}$ permite concatenar dos o más arrays mediante el operador +. Por ejemplo, dados los arrays `a` y `b`, el resultado de su concatenación se muestra en la última línea del siguiente fragmento de código:

```
a = [1, 2, 3]
b = [4, 5, 6]
a + b => [1, 2, 3, 4, 5, 6]
```

donde el símbolo => no es parte del lenguaje, sino que lo hemos usado para indicar cuál es el resultado de una expresión. También es posible obtener un trozo de un array:

```
a = [1, 2, 3, 4]
a[1:3] => [2, 3]
a[1:] => [2, 3, 4]
a[:2] => [1, 2]
```

¿Qué valor tendría entonces `b` después de ejecutar el siguiente código?

```
a = [10, 20, 30, 40]
b = a[2:3] + a[:1]
```

Pregunta 5. La siguiente función recursiva en $\mathcal{L}$ recibe un número entero y un array de enteros (suponemos que ordenados en orden creciente) y devuelve un array de enteros en el que se ha insertado el número entero en la posición correcta. La función, sin embargo, está incompleta: falta la línea con el comentario *missing*.

```
def inserta(num, ordenados):
  if len(ordenados) == 0:
    return [num]
  else:
    if num < ordenados[0]:
      # missing
    else:
      return [ordenados[0]] \
             + inserta(num, ordenados[1:])
```

En lo anterior, `len` es una función que devuelve la longitud de un array. Observa qué devolvería la función tras estas dos llamadas:

```
inserta(10, [-2, 3, 20, 30])
                        => [-2, 3, 10, 20, 30]
inserta(10, []) => [10]
```

¿Qué instrucción debería incluirse en lugar del comentario que reza *missing* para que la función sea correcta?

Pregunta 6. Supongamos definida en el lenguaje $\mathcal{L}$ la función `inserta(num, ordenados)` que devuelve, como en el problema anterior, la inserción ordenada de un número en un array. Por ejemplo:

```
inserta(10, [-2, 3, 20, 30])
                        => [-2, 3, 10, 20, 30]
```

Queremos utilizar la función `inserta` para implementar una función recursiva en $\mathcal{L}$ que reciba un array de números y los devuelva ordenados. Esta es una posible versión incompleta:

```
def ordena(numeros):
  if len(numeros) == 1:
    return numeros
  else:
    # missing
```

Ejemplo de funcionamiento:

```
ordena([10, 8, -1, -10, 2])
                        => [-10, -1, 2, 8, 10]
```

¿Cuál de las siguientes instrucciones debería ir en lugar del comentario que reza *missing* para que `ordena` funcione correctamente? Consulta la pregunta 4 para repasar cómo funcionan los rangos que permiten seleccionar trozos de arrays.

```
a) return ordena(inserta(numeros[0],
                         numeros[1:]))
b) return inserta(numeros[0], numeros[1:])
c) return numeros[0] + ordena(numeros[1:])
d) return inserta(numeros[0],
                  ordena(numeros[1:]))
e) return ordena(numeros[0] + numeros[1:])
f) return ordena(numeros[1:])
```

Pregunta 7. Se disponía del código en lenguaje $\mathcal{L}$ de una función `isAscending` que devuelve True si los elementos de un array de enteros están en orden ascendente y False en otro caso, pero sus líneas se han desordenado y, además, se han mezclado con líneas de otros programas que no estaban en la función original. Indica en el orden adecuado los números de línea que permiten reconstruir la función `isAscending`. Para que veas cuál es el formato que debería tener tu respuesta, una posible respuesta (incorrecta) sería: 01,03,08,09,00.

```
00:   if n >= p:
01:   p = list[0]
02:   if n < p:
03:   def isAscending (list):
04:   p = list[1]
05:   p = n
06:   n = p
07:   for n in list:
08:   return True
09:   return False
```

Pregunta 8. En el lenguaje $\mathcal{L}$ el tiempo de enlace de los métodos tiene que ser declarado explícitamente con las palabras reservadas `virtual` (si el enlace es dinámico, es decir, si la decisión sobre el método específico a invocar se toma en tiempo de ejecución basándose en el tipo efectivo del objeto) o `nonvirtual` (si el enlace es estático, es decir, si la decisión sobre el método concreto a invocar es tomada por el compilador basándose en el tipo del objeto según se conoce en tiempo de compilación). Las siguientes líneas de código definen tres clases diferentes en las que la clase `C` deriva de la clase `B` y esta a su vez de la clase `A`.

```
class A:
  def public virtual f():
    return 2
  def public nonvirtual g():
    return 1

class B extends A:
  def public virtual f():
    return 3
  def public nonvirtual g()
    return 4

class C extends B:
  def public virtual f():
    return 3
  def public nonvirtual g()
    return 5
```

Dadas las definiciones anteriores indica qué valor o valores se imprimen por la función `print` al ejecutar el siguiente código que usa una variable polimórfica `x`:

```
A x
...
x = new C()
int c = 0
while c < 9:
  c = c + x.f() + x.g()
  print(c)
  if x.f() > 2:
    x = new A()
  else:
    x = new B()
```

Pregunta 9. Sea una estructura de datos de tipo árbol donde cada nodo puede tener como máximo dos hijos. Las etiquetas de los nodos son números naturales y no se permiten elementos repetidos (la función para insertar en el árbol retorna silenciosamente sin modificar la estructura de datos si el elemento que se desea insertar ya está en el árbol). Además, los datos se organizan en el árbol de modo que las etiquetas menores que una etiqueta dada se encuentran a la izquierda de dicha etiqueta (subárbol izquierdo) y las mayores se encuentran a la derecha de dicha etiqueta (subárbol derecho). Tras cada inserción, además, el árbol se balancea de modo que las alturas del subárbol izquierdo y del subárbol derecho se diferencien como mucho en 1.

Si en el árbol se han intentado introducir ya 757 números naturales no necesariamente distintos, ¿cuántos nodos es necesario consultar en el peor caso para saber si el número 333 está en el árbol?

Pregunta 10. Se conoce como palabras de Dyck a aquellas cadenas de ceros y unos con el mismo número de ceros y unos en las que ningún subsegmento inicial (es decir, ningún prefijo de la cadena) tiene más unos que ceros. Las siguientes son todas las palabras de Dyck con longitudes 2: 01; las siguientes son todas las palabras de Dyck de longitud 4: 0011, 0101.

La pregunta es: ¿cuántas palabras de Dyck existen de longitud 6?

Pregunta 11. La siguiente función incompleta escrita en lenguaje $\mathcal{L}$ `quita_duplicados` elimina los duplicados de un array y reemplaza los elementos sobrantes por 0.

Por ejemplo, el resultado de quitar duplicados sobre el array [2, 3, 5, 5, 5, 7, 11, 11, 11, 13, 13, 13, 17] es el vector [2, 3, 5, 7, 11, 13, 17, 0, 0, 0, 0, 0, 0].

Escribe la instrucción que debería ir en lugar del comentario que reza *missing* para que funcione correctamente.

```
def quita_duplicados(v):
  write_index = 1
  for i = 1 to len(v) - 1:
    if v[write_index - 1] != v[i]:
      # missing
      write_index = write_index + 1
  for i = write_index to len(v) - 1:
    v[i] = 0
  return v
```