Actas de las Jenui, vol. 8. 2023.
Páginas: 277-286

Modelos mentales erróneos y persistentes en programación*

Francisco J. Gallego-Durán, Patricia Compañ-Rosique, Carlos J. Villagrá-Arnedo
Gala M. García-Sánchez, Rosana Satorre-Cuerda
Smart Learning, grupo de investigación en Tecnologías Inteligentes para el Aprendizaje
Universidad de Alicante
03690 San Vicente del Raspeig (Alicante)
{fjgallego,patricia.company,villagra,galamariagarcia,rosana.satorre}@ua.es

Resumen

El aprendizaje implica la creación de representaciones mentales análogas a lo aprendido, conocidas como modelos mentales. Estos modelos nos permiten realizar predicciones y tomar decisiones. Cuanto más cercanos sean a la realidad, mejores serán nuestras predicciones y decisiones, lo que indica que refinar estos modelos puede representar un aprendizaje efectivo.

La programación, además, requiere realizar el proceso inverso: a partir de los efectos deseados, crear un programa que los produzca. Este proceso, más complejo que simplemente refinar un modelo mental, demanda múltiples modelos precisos y habilidad asociativa para navegar entre ellos en ambas direcciones.

Los profesores enseñamos sobre modelos conceptuales, que son abstracciones claramente definidas. Adquirir estos modelos implica generalizar y extraer patrones de numerosos ejemplos específicos. A menudo, los modelos mentales iniciales se vinculan a detalles circunstanciales de los ejemplos, tanto al ruido como a la señal. Aunque estos modelos sean erróneos, pueden funcionar razonablemente bien en algunas situaciones específicas, ser suficientes para aprobar y persistir en el tiempo. Esto podría explicar dificultades futuras en la programación.

Este estudio busca identificar modelos mentales incorrectos en la programación. Proponemos un método iterativo que parte de hipótesis basadas en la experiencia docente y llega hasta la evidencia empírica. Este enfoque también permite descubrir nuevos modelos y refinar nuestro conocimiento. Presentamos su aplicación con estudiantes de cuarto curso y conceptos de primer curso. Nuestros resultados destacan modelos erróneos potenciales y sugieren otros no previstos que requieren más investigación.

*Este trabajo ha sido financiado por el Instituto de Ciencias de la Educación (ICE) de la Universidad de Alicante a través de su programa de ayudas Redes ICE de Investigación en Docencia Universitaria

Abstract

Learning involves creating mental representations analogous to what is learned, known as mental models. These models allow us to make predictions and decisions. The closer they are to reality, the better our predictions and decisions will be, indicating that effective learning consists of refining these models.

Programming, on the other hand, requires carrying out the reverse process: from the desired effects, create a program that produces them. This process, more complex than merely refining a mental model, demands multiple accurate models and associative skills to navigate between them in both directions.

Teachers instruct on conceptual models, which are clearly defined abstractions. Acquiring these models involves generalizing and extracting patterns from numerous specific examples. Consequently, initial mental models often link to circumstantial details of the examples, both to the noise and the signal. Although these models may be erroneous, they can work reasonably well in some specific situations, be sufficient to pass, and persist over time. This could explain future difficulties in programming.

This study seeks to identify incorrect mental models in programming. We propose an iterative method that starts from hypotheses based on teaching experience and reaches empirical evidence. This approach also allows discovering new models and refining our knowledge. We present its application with fourth-year students and first-year concepts. Our results highlight potential erroneous models and suggest other unforeseen ones that require further investigation.

Palabras clave

Programación, aprendizaje, modelos mentales, educación

1. Introducción

Enseñar y aprender programación son tareas reconocidamente difíciles. Numerosos estudios han documentado los problemas enfrentados por los estudiantes y profesores en este ámbito [24, 5, 4]. Un ejemplo destacado es la encuesta internacional realizada por Lahtinen y colaboradores [14], que incluyó a más de quinientos estudiantes y profesores y resumió las dificultades experimentadas y percibidas en la enseñanza y el aprendizaje de la programación. Este trabajo busca entender las causas fundamentales de estas dificultades y encontrar estrategias para superarlas, mejorar las tasas de éxito y la calidad de los resultados del aprendizaje.

Para abordar estos retos, nos enfocamos en analizar el proceso de aprendizaje. Este proceso comienza con los objetos de aprendizaje, es decir, los modelos conceptuales y las representaciones abstractas de las ideas que componen la teoría de la programación. A partir de estas representaciones, se crean los materiales y explicaciones que los estudiantes utilizan para aprender, induciendo a su vez sus propios modelos mentales.

Nuestro objetivo es ayudar a los estudiantes a desarrollar modelos mentales lo más cercanos posible a los modelos conceptuales. Para lograr esto, nos preguntamos sobre cómo se forman estos modelos mentales, ¿Cómo se forman exactamente los modelos? ¿Son rígidos o maleables? ¿Pueden ser flexibles un tiempo y más rígidos después? ¿Depende del momento en que se forman o de la manera en que se forman? ¿Afecta la edad del aprendiz? ¿Interaccionan unos modelos con otros? Aunque algunas de estas preguntas han sido tratadas en la literatura, aún no disponemos de respuestas detalladas, hasta donde los autores conocemos [11, 13, 18, 20, 21]. Por otra parte, apenas han sido investigadas en el ámbito concreto de la programación. [23]

Además, la programación va más allá de la mera construcción de sólidos modelos mentales. Requiere la capacidad de vincular estos modelos a través de relaciones de causa y efecto, tanto en un sentido directo como inverso. En otras palabras, no solo implica comprender cómo funcionan las diferentes partes de un sistema, sino también ser capaz de visualizar un resultado deseado y retroceder para determinar las acciones o procesos necesarios para lograrlo. Este proceso de 'simulación inversa' es fundamental para la programación efectiva, y requiere un entendimiento profundo y flexible de los modelos mentales subyacentes.

Este estudio da un primer paso para analizar el aprendizaje de la programación desde esta perspectiva, investigando si podemos detectar modelos mentales erróneos en programación y, si es posible, cómo corregirlos o prevenir su aparición. También nos preguntamos si los estudiantes pueden superar evaluaciones con modelos mentales parcialmente funcionales pero incorrectos, y si estos modelos podrían persistir a lo largo del tiempo. Para abordar estas cuestiones, proponemos un método y lo aplicamos a estudiantes de cuarto curso de Ingeniería, buscando modelos mentales erróneos en conceptos de primer curso que puedan seguir presentes en cuarto.

El artículo está estructurado como sigue: el apartado 2 presenta trabajos previos en enseñanza y aprendizaje de programación y modelos conceptuales y mentales; el apartado 3 introduce las preguntas de investigación; el apartado 4 describe el método propuesto; el apartado 5 explica las condiciones experimentales y los resultados obtenidos; el apartado 6 discute las evidencias más relevantes; y, finalmente, el apartado 7 resume las conclusiones obtenidas.

2. Modelos y programación

El aprendizaje implica la construcción de modelos mentales, que son esquemas internos utilizados para representar relaciones entre objetos o representaciones de la realidad, describiendo su funcionamiento [12]. Los estudiantes construyen estos modelos mentales basándose en las explicaciones de los profesores y en los materiales proporcionados, sobre su experiencia y condicionamientos mentales previos. Un modelo conceptual, en cambio, es una representación abstracta, unívoca y simplificada de objetos y sus relaciones en la realidad. Los profesores enseñan modelos conceptuales y los estudiantes construyen sus modelos mentales que son más o menos similares al modelo conceptual explicado por el profesor.

El estudio de la enseñanza y el aprendizaje de la programación de ordenadores ha despertado un gran interés en la relación entre modelos mentales y conceptuales. Pirolli y colaboradores [19] realizaron dos experimentos que indican que proporcionar a los estudiantes representaciones comprensibles de programas y guiarlos a través de una estrategia ideal de resolución de problemas facilita el aprendizaje.

Mazumder y colaboradores [17] observaron que los programadores novatos a menudo tienen modelos mentales inconsistentes y defectuosos de conceptos básicos de programación. Desarrollaron un cuestionario de opción múltiple para medir la consistencia del modelo mental de un programador novato sobre arrays.

Javed y colaboradores [9] destacaron que un principiante construye un modelo mental de la ejecución de un programa mientras aprende a programar. Cualquier malentendido en esta etapa puede conducir al desarrollo de un modelo mental incorrecto. Para prevenir esta situación, utilizaron el *Little Man Computer (LMC)*, una herramienta interactiva visual que ayuda a los estudiantes a comprender cómo el software interactúa con

el hardware para lograr el objetivo del programador. Tras trabajar con LMC, los estudiantes pasaron a utilizar *Scratch*, donde se mapearon las construcciones de programación equivalentes en LMC.

Almadhoun y colaboradores [1] estudiaron la precisión de los modelos mentales de los principiantes en informática sobre las listas enlazadas. Realizaron entrevistas semi-estructuradas con once estudiantes universitarios y descubrieron que ninguno tenía un modelo mental preciso de las listas enlazadas simples, incluso después de haber aprendido sobre ellas y aplicado ese conocimiento en un curso de estructuras de datos.

Henry y colaboradores [8] desarrollaron una evaluación basada en un inventario de conceptos para perfilar a los estudiantes según su comprensión de conceptos específicos de programación. Su objetivo era identificar modelos mentales incorrectos que pudieran obstaculizar el progreso de los estudiantes, lo que les permitió definir nueve perfiles diferentes.

El estudio de los modelos mentales de los estudiantes en el ámbito de la recursividad también ha sido objeto de investigación. Law [15] se centró en la dificultad que enfrentan los programadores principiantes al construir un modelo mental adecuado del concepto de recursividad, específicamente en relación con la pila del programa y el flujo de control hacia atrás desde la última invocación del método recursivo. Para abordar esta dificultad, Law utilizó una interfaz visual que no requería que los programadores novatos aplicaran los pasos de la solución mediante un lenguaje de programación específico. Al ofrecer un enfoque independiente del lenguaje de programación, los programadores novatos pudieron comprender el concepto de recursividad de manera que pudieran desarrollar un algoritmo aplicable en cualquier lenguaje de programación.

Chao y colaboradores [2] basaron su investigación en el marco del Conocimiento en Piezas (*KiP*). Este marco teórico sostiene que los estudiantes poseen una variedad de piezas o fragmentos de conocimiento, que no están necesariamente organizados o conectados entre sí. Estos fragmentos pueden ser activados y aplicados de manera dinámica y contextual, lo que da cuenta de la variabilidad y la adaptabilidad del conocimiento humano. La aplicación del marco *KiP* a la enseñanza de la programación permite una comprensión más matizada de cómo los estudiantes utilizan y adaptan sus modelos mentales a medida que aprenden a programar. Chao y colaboradores [2] clasificaron los resultados de su estudio de acuerdo con una clasificación de modelos mentales de recursividad que ellos mismos propusieron. Estos enfoques no son nuevos y han sido explorados en estudios anteriores con diferentes esquemas de aplicación pero con objetivos y resultados similares [31, 6, 7].

La enseñanza de la programación es considerada una disciplina difícil, ya que programar requiere habilidades lógico-matemáticas, capacidad de abstracción y resolución de problemas, entre otras. Con el objetivo de que los modelos mentales construidos por los estudiantes sean lo más similares posibles a los modelos conceptuales enseñados por los profesores, los docentes de esta disciplina introducen cambios en su metodología año tras año para tratar de mejorar el aprendizaje de sus estudiantes. A continuación resumimos algunas de las más relevantes.

En cuanto a metodologías de enseñanza de programación, Satorre y colaboradores [28] presentaron un enfoque para fundamentos de programación en la que los docentes intentan que los estudiantes aprendan a pensar, a analizar situaciones y a diseñar el método de resolución más adecuado, sin centrarse en un lenguaje de programación específico. Por su parte, Compañ-Rosique y colaboradores [3] analizaron cómo afectan al rendimiento de los estudiantes las diferentes metodologías docentes durante varios cursos académicos, resultando que una metodología híbrida (conferencias y aprendizaje invertido) parece ser más efectiva.

Otros autores han incorporado tecnologías y técnicas diversas para evaluar su impacto en el rendimiento de los estudiantes. Özyurt y colaboradores [32] evaluaron el efecto del uso de *Facebook* para mejorar el aprendizaje de los estudiantes en programación. Concluyeron que aunque el uso de *Facebook* puede causar distracción y disminuir la comunicación cara a cara, también aumenta la comunicación extracurricular y la motivación. Pérez-Carrasco y colaboradores [22] implementaron una aplicación que ofrecía soporte para la generación automatizada de visualizaciones de programas con posibilidad de animación, destinado principalmente a la comprensión de la recursividad. Salazar Mesía y colaboradores [26] y Sittiyuno y Chaipah [30] describen experiencias de implementación de material educativo basado en realidad aumentada en asignaturas de programación.

Scott y colaboradores [29] realizaron un curso de introducción a la programación en el que se le proporcionó a cada estudiante un robot programable para aprovechar una experiencia atractiva, así como la retroalimentación inmediata e intuitiva. Además, los estudiantes participaban en unas Olimpiadas de la Robótica, lo que les motivaba tanto para practicar con mayor frecuencia como para mejorar la calidad de su código. Marco-Galindo y colaboradores [16] se centraron en los estudiantes repetidores, con el objetivo de diseñar una estrategia que les ofrezca una formación adaptada a su situación específica.

La enseñanza de la programación en disciplinas que no son la Ingeniería Informática presenta desafíos únicos que requieren enfoques de enseñanza diferenciados. Rubio Escudero y colaboradores [25] utilizaron la

computación física, que introduce los conceptos de la programación en el mundo real para que el estudiante interactúe con ellos. Hicieron uso de plataformas *Arduino* y *LEGO* para enseñar a estudiantes del grado de Biología y de Matemáticas. Así mismo, Sales y colaboradores [27] implantaron en asignaturas de ingenierías de ámbito industrial micro-talleres de programación en grupo con *Arduino*, para fomentar la comprensión de la materia. Finalmente, Jiménez-García y colaboradores [10] describen la experiencia de impartir programación en dos asignaturas de nivel de Máster en bioinformática y ciencia de datos.

En resumen, existen numerosos estudios sobre la teoría de Modelos Mentales, y también muchísimos estudios sobre metodologías y distintas aproximaciones didácticas y tecnológicas a la enseñanza de la programación. Sin embargo, hasta donde los autores tenemos conocimiento, hemos podido encontrar muy pocos estudios que tengan en cuenta ambas cuestiones simultáneamente. Así pues, este trabajo pretende contribuir y promover la aplicación de la teoría de Modelos Mentales en el análisis y mejora de la enseñanza de la programación, pues consideramos que hay mucho potencial por explorar.

3. Preguntas de Investigación

Este estudio se centra en las siguientes preguntas de investigación:

- RQ1: ¿Cómo podemos identificar la presencia de modelos mentales erróneos en los estudiantes de programación?
- RQ2: ¿Puede un modelo mental erróneo ser suficiente para permitir a los estudiantes superar evaluaciones y perdurar en el tiempo?

Para responder a la primera pregunta (RQ1), proponemos un método de estudio y realizamos un experimento con los estudiantes de cuarto curso de Ingeniería Informática. El objetivo es desarrollar y probar un enfoque sistemático para detectar y describir modelos mentales erróneos en el contexto de la programación.

En cuanto a la segunda pregunta (RQ2), partimos de la hipótesis de que es cierta, basándonos en la experiencia docente de los profesores. Los profesores han observado que algunos estudiantes superan las evaluaciones pese a tener conceptos modelados erróneamente. Por tanto, buscamos evidencias que validen la existencia en cuarto curso de modelos mentales erróneos sobre conceptos de primero.

Es importante aclarar que esto no constituye una prueba definitiva de que dichos modelos hayan sido usados en procesos anteriores de evaluación, ni de que hayan perdurado en el tiempo; podrían haberse formado recientemente, habiendo existido otros previamente. Así pues, consideraremos la existencia de modelos erróneos en cuarto curso, sobre conceptos que han sido evaluados previamente múltiples veces, un indicio fuerte para apoyar la realización de estudios longitudinales que puedan validar RQ2. En ese sentido, este trabajo pretende humildemente ser un primer paso.

4. Método propuesto

Proponemos un método iterativo en cinco pasos para obtener evidencia sobre los modelos mentales de los estudiantes. El método partiría de la experiencia previa de los profesores: su conocimiento de los modelos conceptuales de la materia y las observaciones anecdóticas en su día a día de clase. Esta sería la semilla para iniciar el primer paso de la primera iteración, siguiendo este esquema:

1. Describir modelos mentales hipotéticos.
2. Diseñar preguntas de prueba.
3. Realizar un cuestionario.
4. Extraer evidencias de las respuestas.
5. Clasificar y seleccionar modelos y preguntas.
6. Repetir.

4.1. Describir modelos hipotéticos

Comenzamos por conceptualizar y describir los posibles modelos mentales que los estudiantes podrían tener. Un modelo mental, en este contexto, se refiere a la manera en la que los estudiantes entienden y conceptualizan un aspecto o concepto de programación. Por ejemplo, un modelo mental común puede ser .[El] estudiante considera que las variables son simples contenedores de datos". Esta concepción, aunque útil en los niveles iniciales, puede llevar a confusiones cuando se trabaja con conceptos más avanzados como referencias o punteros.

Con la experiencia docente como punto de partida, los profesores generan *hipótesis* sobre otros posibles modelos mentales erróneos que los estudiantes pueden albergar. Se elabora una lista describiendo las características de cada modelo mental hipotético. Un ejemplo de estos modelos podría ser “El estudiante piensa que el tipo *char* sólo puede almacenar caracteres”. Esto indicaría una comprensión errónea de este tipo de dato, que en realidad almacena enteros en 8 bits (los caracteres son simplemente una representación de esos números de acuerdo con el código ASCII).

Este primer paso de describir modelos mentales hipotéticos es crucial para comenzar a entender cómo los estudiantes conceptualizan los conceptos de programación y dónde pueden estar surgiendo los malentendidos.

4.2. Diseñar preguntas de prueba

Elaborada la lista de modelos hipotéticos, se *diseñan preguntas* para medir su existencia. Una buena pregunta sería la que permitiera discriminar la presencia o no de un modelo mental erróneo. Para esto, es conveniente proyectar bien cómo piensa un estudiante con un modelo concreto, y diseñar una cuestión donde podamos predecir su respuesta. Es importante tener en cuenta que este diseño es complejo y puede haber muchos factores influyentes. Por este motivo, las propias preguntas deberán después someterse a juicio para saber si están funcionando adecuadamente para su propósito.

Consideramos clave diseñar preguntas de respuesta abierta. La tendencia natural es a simplificar, creando preguntas cerradas, que den información específica de acierto/fallo, sí o no. Sin embargo, buscamos explorar los procesos mentales del estudiante. Para esto es muy relevante contar con sus reflexiones y explicaciones. Con preguntas cerradas, algunas respuestas podrían estar disociadas del proceso mental, podrían ser casuales. Esta situación es menos probable cuando se piden explicaciones, argumentos o reflexiones. Además, se obtiene un beneficio clave: un estudiante que no tenga un modelo mental hipotético, puede desvelar en su respuesta otros modelos mentales erróneos no previstos. Esta información es posiblemente más valiosa que identificar si tiene un modelo. Con ella podemos identificar nuevos patrones e hipotetizar sobre otros modelos mentales erróneos a estudiar en siguientes iteraciones. Así, a lo largo de distintas iteraciones, se puede ir descubriendo y afinando muchos modelos mentales erróneos de estudiantes.

4.3. Realizar un cuestionario

Para realizar el cuestionario es importante disponer de condiciones controladas. Las reflexiones de los estudiantes deben ser lo más crudas posible para acercarnos a lo que piensan. Son especialmente relevantes cuando están equivocados. Por este motivo, recomendamos una introducción previa de concienciación: transmitir a los estudiantes que no es un examen, sino un ejercicio para poder ayudarles a mejorar. Indicarles que lo más valioso son sus reflexiones, y que son más valiosas aún si están equivocados. Hacerles ver que no sólo no deben tener vergüenza o miedo a equivocarse, que sus equivocaciones son precisamente lo que nos permitirá ayudarles a mejorar. Sin una concienciación como esta, es probable que los estudiantes actúen como en un examen, evitando reflexiones que les dejen en evidencia o no respondiendo preguntas.

Así mismo, recomendamos que sea presencial y que se utilicen medios que les eviten la tentación de buscar respuestas. Cuestionario en papel o a través de *Moodle* con *Safe Exam Browser* pueden ser buenas opciones.

4.4. Extraer evidencias de las respuestas

Consiste en analizar cada respuesta y deducir si evidencia alguno de los modelos mentales propuestos. En una matriz, se disponen respuestas en filas y modelos en columnas. Cada celda cruza una respuesta (fila) con un modelo mental (columna) y nos permite clasificar como (1) el estudiante piensa como en el modelo o (0) no podemos deducir que piense como el modelo. Además de esta clasificación para preguntas y modelos, también podemos evaluar la respuesta a la pregunta. Proponemos hacerlo con 0 (no se responde) y de 1 a 5, graduando desde totalmente errónea a totalmente correcta. Esta evaluación permitirá también correlacionar grado de acierto de los estudiantes con modelos mentales asociados.

Con esta sencilla, pero laboriosa, clasificación se pueden evaluar modelos y preguntas simplemente sumando los unos. A mayor número de unos, más prevalente es un modelo, y también más reveladora es una pregunta.

Como indicábamos antes, además de la cuantificación, disponer de respuestas abiertas ofrece la posibilidad de identificar nuevos patrones y modelos. Durante el análisis es importante tomar anotaciones de los errores de razonamiento, consideración y terminología. Con estas anotaciones se podrá encontrar patrones que permitirán lanzar nuevas hipótesis sobre modelos mentales erróneos. Estas nuevas hipótesis se incorporarán al primer paso de la siguiente iteración.

4.5. Clasificar, seleccionar y reiterar

Partiendo del número de estudiantes que evidencian cada modelo, y del número de modelos que evidencian cada pregunta, clasificamos modelos y preguntas según su relevancia. Consideramos el dato de cada modelo como una muestra de una distribución normal y calculamos su media y desviación típica. Así podemos clasificarlos en relevancias: (Alta) superan la media más la desviación típica, (Baja) no superan la media menos la desviación típica, (Media) el resto. Con las preguntas podemos hacer la misma clasificación.

Una vez clasificados modelos y preguntas, podemos seleccionar para iterar. Podemos mantener sin cambios los de relevancia alta, eliminar los de relevancia baja y reconsiderar los que tengan relevancia media. Al reconsiderar se debería analizar modelos y derivar de ellos nuevas versiones de ambos. El objetivo debería ser comprobar si los modelos requieren definirse mejor, o si se debe preguntar de otra manera para evidenciar mejor los modelos.

Con todo, se obtiene una nueva lista de modelos mentales y de preguntas actualizadas. Sería el momento de volver al primer paso y repetir.

5. Experimentación

Siguiendo el método propuesto, los profesores de la Universidad de Alicante realizamos una primera iteración completa para este estudio. Comenzamos seleccionando modelos conceptuales en los que observamos problemas, y elaborando la lista de modelos mentales erróneos hipotéticos y preguntas para discriminar su existencia[1]. Configuramos y administramos el cuestionario a 50 estudiantes de cuarto curso del Grado en Ingeniería Multimedia (12 chicas, 38 chicos) con edades entre 22 y 25 años. Los datos crudos de respuestas anonimizadas, y los análisis completos presentados resumidamente en este apartado, están publicados como material adicional[1].

El cuestionario constaba de 30 preguntas: 24 sobre programación en C++, una calificación personal sobre el gusto por programar (de 0 a 10), cuatro referentes a hábitos de sueño y una de consentimiento informado. Las preguntas de programación se puntuaron de 1 (totalmente incorrecto, entendido como un modelado erróneo) a 5 (totalmente correcto, indicando una comprensión acorde con el modelo conceptual), sin tener en cuenta la claridad de expresión escrita del estudiante. Para cada respuesta, identificamos y clasificamos los modelos mentales que se evidenciaban. Agrupamos las preguntas de programación por temática en: variables, tipos de datos, el tipo *char*, punteros, parámetros y memoria; y propusimos 27 modelos mentales hipotéticos. Aún tratando de minimizarlo, este tipo de evaluación es inevitablemente subjetivo, lo que debe ser considerado al sopesar el valor de la evidencia obtenida.

5.1. Resultados analizados

En primer lugar estudiamos la alineación entre preguntas, nota media de las respuestas (evaluación) y cuánto les gusta programar a los estudiantes. Se espera que quienes gusten más de programar obtengan mejor resultado, indicando tanto un lógico mayor desarrollo de su comprensión como un apropiado diseño y evaluación del cuestionario. La figura 1 muestra la relación y una tendencia a mejor evaluación cuanto más gusta programar, con estadísticos de correlación *Pearson* $\rho = 0{,}517$ y *Spearman* $\rho = 0{,}524$.

A continuación analizamos cuántos modelos mentales erróneos han sido identificados en los estudiantes. La figura 2 muestra un histograma de frecuencias en el que se aprecia que 36 de los 50 estudiantes (72 %) evidencian entre 10 y 19 modelos mentales erróneos, según nuestro análisis. Así mismo, tan sólo un estudiante evidencia tener menos de 5 modelos mentales

[1] `http://hdl.handle.net/10045/134282`. Último acceso 12/05/2023.

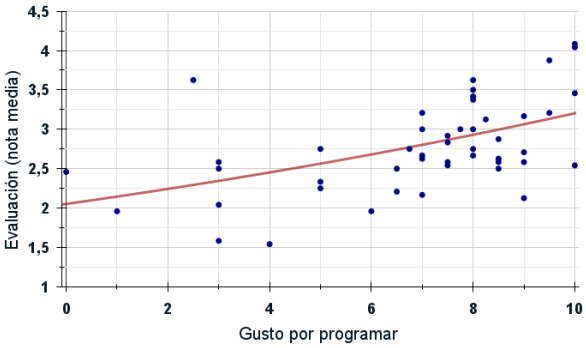

Figura 1: Relación entre gusto por programar y nota

(3 en concreto), siendo 23 el máximo por estudiante. La distribución es normal con $(\mu, \sigma) = (14{,}58; 4{,}63)$.

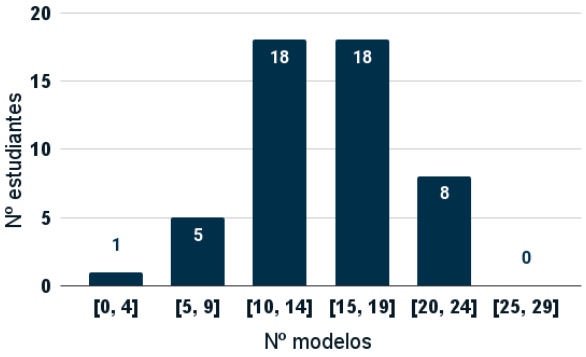

Figura 2: Modelos mentales erróneos por estudiante

Siguiendo el quinto paso del método propuesto, clasificamos modelos y métodos para su selección previa a una siguiente iteración. La figura 3 muestra la distribución de los 27 modelos estudiados según su incidencia analizada (número de apariciones en estudiantes). Se asume distribución normal y se calculan los descriptores $(\mu, \sigma) = (27; 16{,}35)$. La figura 3 muestra sombreado el rango esperado para el 68 % de las muestras, $[\mu - \sigma, \mu + \sigma]$. Se encuentran 6 modelos con incidencia baja ($< \mu - \sigma$) y 3 con incidencia alta ($> \mu + \sigma$). Los modelos con incidencia alta son: "[M13] Creer que caracteres y valores enteros son objetos diferentes", "[M15] Pensar que un puntero *ES* el objeto al que apunta" y "[M19] Pensar que los parámetros no son variables locales (como si fueran un tipo especial de objetos)".

Respecto a los análisis cualitativos, destacamos algunos patrones encontrados en las respuestas, que sugieren nuevos modelos mentales erróneos. Los patrones más relevantes son:

- Pensar que para obtener la dirección de memoria de una variable se necesita un puntero (y no simplemente usar el operador dirección &).
- Confundir los operadores & y *.

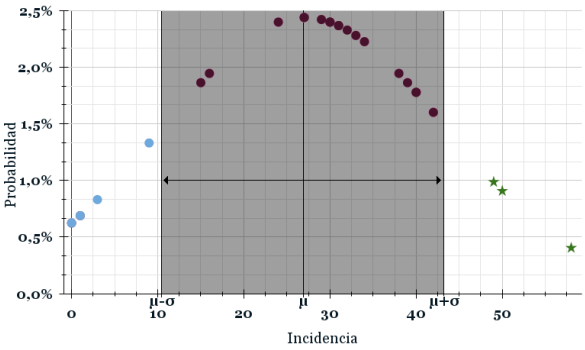

Figura 3: Clasificación de los modelos estudiados

- Confundir y mezclar reserva/liberación con creación/destrucción e inicialización/borrado.
- Creer que la pila de programa o el *heap* no están en la memoria.
- Creer que los punteros no son variables.
- Creer que una variable *int* no puede almacenar caracteres.
- Creer que de algunas variables se puede obtener su dirección de memoria y de otras no.

6. Discusión

En una primera consideración, los resultados obtenidos parecen ser razonables a nivel numérico. Existe una correlación previsible entre cuánto disfrutan los estudiantes la programación y sus evaluaciones medias. En la figura 1 se percibe visualmente la correlación y la similitud entre los coeficientes de Pearson y Spearman sugiere que es bastante lineal. Aunque la correlación es moderada (alrededor de $0{,}52$), esto sugiere que las preguntas y la evaluación son fuentes apropiadas de evidencia.

Iniciamos la investigación con el objetivo de identificar modelos mentales erróneos. La figura 3 nos indica que 3 de los modelos hipotéticos tienen una incidencia destacable en el grupo de estudiantes, siendo clasificados como relevancia alta. Estos modelos, además, se corresponden con errores habituales que los profesores reconocemos en la experiencia docente cotidiana y que causan problemas. Consideramos esta evidencia numérica un indicio que sugiere la presencia de esta forma concreta de pensar en los estudiantes. Este indicio se ve con más claridad en las respuestas de los estudiantes. Por ejemplo, la pregunta Q12 les propone analizar este código:

```
char probemos() {
char a='A'; char b=65; return a-b; }
```

Y los estudiantes dan respuestas como:

- "*65 no es un char*, por lo que daría errores"
- "la resta *no se puede hacer* entre char y número"
- "ambas *simbolizan una cadena* que contienen caracteres *de un abecedario*, *no sus valores*, por lo cual b serían los digitos 6 y 5 y no el valor numérico 65."
- "Esta funcion podra tener comportamiento no definido, ya que restaras *el valor al que apunte a* para interpretar el caracter A y a eso se le restará 65, lo cual puede dar un numero negativo y de ser asi devolveria lo que el compilador interpretase"

Entre todas las preguntas realizadas, hemos encontrado hasta 50 respuestas con estos errores referidos al modelo mental M13, donde creen que caracteres y números son objetos diferentes. De hecho, muy pocos indican en sus respuestas que 'A' no es una "letra" sino un código ASCII, es decir, un número entero (un int, de hecho).

Por su parte, el modelo mental "[M15] Pensar que un puntero *ES* el objeto al que apunta" está infrarepresentado debido a nuestro análisis segmentado. Sólo está considerado en las preguntas sobre punteros, pero también ha aparecido en las preguntas sobre memoria. Por ejemplo, la pregunta Q27 mostraba un código similar (comprimido por espacio),

```
struct P{int x, y;}; int main() {
P* p1=new P{1,2}; P* p2=new P{2,3};
P* a=p1; delete a; a=p2; delete a;
/*..............*/ }
```

Y preguntaba si hay errores y si se puede modificar el objeto P{1,2} en "/*...*/"(línea 9). Algunas respuestas:

- "en la linea 8 el ordenador no sabe que es a.[...]*no puedes reescribir en esa dirección de memoria y eliminarla de nuevo*"
- "al *eliminar la variable* a en la linea 7, después no puedes modificar nada ya que es comportamiento no definido"
- "lo que *se ha eliminado son variables* que apuntan a su direccion de memoria no el puntero p1."
- "si se puede modificar el Point{1,2} de la linea 9, porque aunque sea haga que a = p1 y *se borre a*, p1 sigue existiendo"
- "error en la línea 8, porque en ese momento [...] *a no existe* y para acceder a ella habría que crearla de nuevo antes de asignarle valor directamente."

Muchas respuestas similares identifican que no disocian variable puntero y objeto apuntado como entidades independientes. Esto refuerza el modelo M15. Además, añade matices interesantes para investigar en futuras iteraciones, pues se aprecian distintas formas de interpretar esta situación.

Los resultados aportan muchas observaciones en estas líneas, lo que nos hace pensar que con este método podemos identificar modelos mentales erróneos, concretarlos y analizarlos y perfilarlos. Siendo conscientes

de que la muestra es pequeña y es un primer experimento que requiere refrendo por otros estudios, consideramos que el método puede ser útil y proponemos su uso para recabar más evidencia y comprobar si apoya esta línea.

En cuanto a si un modelo mental erróneo puede persistir a pesar de las evaluaciones y con el tiempo, nuestros resultados sugieren que esto puede ser posible. Siendo estrictos, no tenemos datos para afirmar si los modelos mostrarían evidencias similares al terminar primer curso que en la actualidad. Sin embargo, entendemos razonable la hipótesis de que así sea, puesto que los estudiantes han estado programando durante 3 años y la experiencia entendemos que debería haberles hecho mejorar. No parece probable que al terminar primer curso tuvieran modelos mentales más aproximados a los conceptuales que se hayan deteriorado con el tiempo. Aún así, no podemos descartar esta posibilidad hasta que obtengamos evidencias.

Sin embargo, podemos afirmar con certeza que los estudiantes exhiben modelos mentales de pensamiento que no son deseables ni esperados en el cuarto año de ingeniería, después de más de 3 años de programación. Son problemas en conceptos básicos que deben manejar continuamente, con los que construyen todo. Es esperable, por tanto, que estos errores se propaguen por sus razonamientos, diseños y desarrollos y produzcan problemas, frustraciones y dificultades en sus proyectos. Podrían incluso motivar sus estrategias de comportamiento en el desarrollo, en las que tienden a buscar por Internet tutoriales y código de partida en el que apoyarse, en vez de crearlo por ellos mismos. ¿Podríamos hacerles ganar seguridad corrigiendo estos modelos? Podría ser interesante considerar cuestiones similares en futuros estudios.

6.1. Limitaciones

Este estudio, aunque ofrece indicios valiosos, se encuentra limitado por varios factores. La muestra es pequeña y única, compuesta por 50 estudiantes de Ingeniería Multimedia de la Universidad de Alicante, lo cual puede limitar la extrapolación de los resultados a otros contextos o universidades. Además, los modelos y preguntas utilizados son los primeros de su tipo, lo que puede introducir sesgos no previstos.

Es importante también tener en cuenta la subjetividad inherente a la evaluación abierta utilizada en este estudio. La interpretación de las respuestas de los estudiantes y la asignación de modelos mentales conlleva un cierto grado de juicio por parte del evaluador, lo que puede afectar a los resultados. A pesar de estas limitaciones, creemos que los hallazgos representan un punto de partida valioso para futuras investigaciones.

7. Conclusiones

Este trabajo parte de una línea general de preocupación por la enseñanza y el aprendizaje de la programación en las carreras de ingeniería. Nos planteamos el análisis de los problemas de los estudiantes desde el paradigma de los modelos conceptuales y los modelos mentales. También, nos preguntamos si podemos identificar modelos mentales erróneos en los estudiantes y pueden superar evaluaciones con esos modelos erróneos, haciéndolos perdurar en el tiempo. Proponemos un modelo iterativo en cinco pasos para obtener evidencia sobre estos modelos erróneos. El modelo se centra en diseñar preguntas discriminatorias de respuesta abierta que estén diseñadas específicamente para poner a prueba modelos hipotéticos. Planteamos también una forma de analizar los resultados, seleccionar y continuar en futuras iteraciones.

Aplicamos el modelo propuesto en un experimento con estudiantes de cuarto curso del grado en Ingeniería Multimedia de la Universidad de Alicante. Modelos conceptuales, mentales hipotéticos, preguntas, respuestas obtenidas y análisis están publicados como material adicional[1]. Tras analizar, obtenemos indicios de que algunos de nuestros estudiantes razonan siguiendo algunos de los modelos mentales propuestos. Además, los análisis cualitativos nos llevan también a proponer nuevos modelos mentales no previstos, que podrán ser estudiados en futuras iteraciones. Estos modelos proceden de patrones observados en las respuestas abiertas de los estudiantes.

Los indicios de modelos mentales erróneos en nuestros estudiantes se refieren a conceptos de programación de primer curso. Esto nos lleva a concluir también que los estudiantes podrían superar evaluaciones teniendo algunos modelos mentales erróneos, y que estos modelos perdurarían en el tiempo.

Pese a las limitaciones por ser un primer estudio, creemos que este método propuesto representa un punto de partida de investigación con potencial, a tenor de los indicios encontrados. Continuaremos utilizando el método para obtener más evidencia, en particular observando la evolución de los estudiantes de primer a cuarto curso. Así mismo, procederemos a refinar modelos y preguntas para intentar mejorar la calidad de la evidencia y la concreción de los modelos. Sin embargo, estimamos muy necesario que otros investigadores en distintas universidades, grados y países puedan obtener evidencia para entender mejor qué conclusiones son generalizables y cuáles no, y en qué medida. Así pues, animamos a otros investigadores a repetir el experimento y compartir evidencias.

Referencias

[1] Eman Almadhoun y Jennifer Parham-Mocello. Exploratory Study on Accuracy of Students' Mental Models of a Singly Linked List. En *2021 IEEE Frontiers in Education Conference (FIE)*, páginas 1–9, octubre 2021. ISSN: 2377-634X.

[2] Jie Chao, David F. Feldon y James P. Cohoon. Dynamic Mental Model Construction: A Knowledge in Pieces-Based Explanation for Computing Students' Erratic Performance on Recursion. *Journal of the Learning Sciences*, 27(3):431–473, julio 2018. Publisher: Routledge _eprint: https://doi.org/10.1080/10508406.2017.1392309.

[3] Patricia Compañ-Rosique, Rafael Molina-Carmona y Rosana Satorre-Cuerda. Effects of Teaching Methodology on the Students' Academic Performance in an Introductory Course of Programming. En Panayiotis Zaphiris and Andri Ioannou, editores, *Learning and Collaboration Technologies. Designing Learning Experiences*, Lecture Notes in Computer Science, páginas 332–345, Cham, 2019. Springer International Publishing.

[4] Jorge Iván Fuentes-Rosado y Melquizedec Moo-Medina. Dificultades de aprender a programar. *Revista Educación en Ingeniería*, 12(24):76–82, agosto 2017.

[5] Francisco J. Gallego-Durán, Rosana Satorre-Cuerda, Patricia Compañ-Rosique, Carlos J. Villagrá-Arnedo, Rafael Molina-Carmona y Faraón Llorens-Largo. A low-level approach to improve programming learning. *Universal Access in the Information Society*, 20(3):479–493, agosto 2021.

[6] David Ginat y Eyal Shifroni. Teaching recursion in a procedural environment—how much should we emphasize the computing model? En *The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, páginas 127–131, New Orleans Louisiana USA, marzo 1999. ACM.

[7] Katherine Gunion, Todd Milford y Ulrike Stege. Curing recursion aversion. *ACM SIGCSE Bulletin*, 41(3):124–128, agosto 2009.

[8] Julie Henry y Bruno Dumas. Developing an Assessment to Profile Students based on their Understanding of the Variable Programming Concept. En *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, páginas 33–39, New York, NY, USA, junio 2020. Association for Computing Machinery.

[9] Noman Javed y Faisal Zeeshan. LMC + Scratch: a recipe to construct a mental model of program execution. En Rosanne English and Craig Stewart, editores, *6th Conference on Computing Education Practice, CEP 2022*, páginas 33–36. Association for Computing Machinery, New York, NY, enero 2022.

[10] Brian Jiménez-García, Cristina Pérez Solà, Pau Andrio Balado y Maria-Jesús Marco-Galindo. Aprendiendo a programar. Nuevos retos, nuevas propuestas. En *Actas de las XXV Jornadas de Enseñanza Universitaria de Informática, JENUI 2019*, páginas 71–78, Murcia, julio 2019.

[11] Phillip Nicholas Johnson-Laird. *Mental Models: Towards a Cognitive Science of Language, Inference y Consciousness*. Harvard University Press, USA, 1986.

[12] Rosária Justi y Jan Driel. The use of the interconnected model of teacher professional growth for understanding the development of science teachers' knowledge on models and modelling. *Teaching and Teacher Education*, 22:437–450, mayo 2006.

[13] Annette Karmiloff-Smith. *Beyond Modularity: A Developmental Perspective on Cognitive Science*. The MIT Press, 1995.

[14] Essi Lahtinen, Kirsti Ala-Mutka y Hannu-Matti Järvinen. A study of the difficulties of novice programmers. En *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, ITiCSE '05, páginas 14–18, New York, NY, USA, junio 2005. Association for Computing Machinery.

[15] Robert Law. Introducing Novice Programmers to Functions and Recursion Using Computer Games. *European Conference on Games Based Learning*, pages 325–334, 2018. páginas: 325-334 Place: Reading, United Kingdom Publisher: Academic Conferences International Limited.

[16] Maria-Jesús Marco-Galindo, Julià Minguillón, David García-Solórzano y Teresa Sancho-Vinuesa. ¿Quién tropieza dos veces con la misma piedra en una asignatura inicial de programación? En *Actas de las XXVII Jornadas de Enseñanza Universitaria de Informática, JENUI 2021*, páginas 59–66, Valencia, julio 2021.

[17] Syeda Fatema Mazumder y Manuel A. Pérez-Quiñones. Eliciting A Novice Programmer's Mental Model of Arrays. En *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, página 1302, New York, NY, USA, marzo 2021. Association for Computing Machinery.

[18] Donald A. Norman. Some observations on mental models. En *Mental models*, páginas 15–22. Psychology Press, 2014.

[19] Peter L. Pirolli y John R. Anderson. The Role of Mental Models in Learning to Program. Techni-

cal report, N/A, noviembre 1984. ERIC Number: ED265177.

[20] Novi Prayekti, Toto Nusantara, Sudirman y Hery Susanto. Students' mental model in solving the patterns of generalization problem. *IOP Conference Series: Earth and Environmental Science*, 243:012144, abril 2019.

[21] Novi Prayekti, Toto Nusantara, Sudirman Sudirman, Hery Susanto y Imam Rofiki. Students' mental models in mathematics problem-solving. *Journal of critical reviews*, 7(12), junio 2020.

[22] Antonio Pérez-Carrasco y Jesús Ángel Velázquez Iturbide. Animación automatizada de técnicas de diseño de algoritmos. En *Actas de las XV Jornadas de Enseñanza Universitaria de Informática, JENUI 2009*, páginas 115–122, Barcelona, julio 2009.

[23] Anthony Robins. Learning edge momentum: a new account of outcomes in CS1. *Computer Science Education*, 20(1):37–71, 2010.

[24] Anthony Robins, Janet Rountree y Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.

[25] Miguel Ángel Rubio Escudero, Carolina Mañoso Hierro, Rocío C. Romero Zaliz, y Angel Pérez de Madrid. Uso de las plataformas LEGO y Arduino en la enseñanza de la programación. En *Actas de las XX Jornadas sobre la Enseñanza Universitaria de la Informática: JENUI 2014*, páginas 419–426. Universidad de Oviedo, 2014.

[26] Natali Salazar Mesía, Cecilia Verónica Sanz y Gladys Gorga. Experiencia de enseñanza de programación con Realidad Aumentada. En *Actas de las XXII Jornadas sobre la Enseñanza Universitaria de la Informática: JENUI 2016*, 2016.

[27] Jorge Sales, Gabriel Recatalá y José V. Martí. Micro-talleres Arduino: una experiencia piloto en informática para ingenierías en el aprendizaje de la programación. En *Actas de las XXI Jornadas de Enseñanza Universitaria de Informática, JENUI 2015*, páginas 302–309 Andorra la Vella: Universitat Oberta La Salle, julio 2015.

[28] Rosana Satorre Cuerda, Patricia Compañ Rosique y Faraón Llorens Largo. Un modo de entender la programación. En *X Jornadas de Enseñanza Universitaria de la Informática: JENUI 2004.*, páginas 387–392. Thomson-Paraninfo, 2004.

[29] Michael James Scott, Steve Counsell, Stanislao Lauria, Stephen Swift, Allan Tucker, Martin Shepperd y Gheorghita Ghinea. Enhancing practice and achievement in introductory programming with a robot olympics. *IEEE Transactions on Education*, 58(4):249–254, 2015.

[30] Sirawit Sittiyuno y Kornchawal Chaipah. Arcode: Augmented reality application for learning elementary computer programming. En *16th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, páginas 32–37, 2019.

[31] Cheng-Chih Wu, Nell B. Dale y Lowell J. Bethel. Conceptual models and cognitive learning styles in teaching recursion. En *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education - SIGCSE '98*, páginas 292–296, Atlanta, Georgia, United States, 1998. ACM Press.

[32] Özcan Özyurt y Hacer Özyurt. Using facebook to enhance learning experiences of students in computer programming at introduction to programming and algorithm course. *Computer Applications in Engineering Education*, 24(4):546–554, 2016.