El efecto de la teoría de carga cognitiva en el aprendizaje de la programación básica

The Effect of Cognitive Load Theory on the Learning of Basic Programming

ISSN 2071-8748
E-ISSN 2218-3345

URI: http://hdl.handle.net/11298/963
DOI: https://doi.org/10.5377/entorno.v0i67.7500

Carlos Argelio Arévalo-Mercado[1]
ORCID: 0000-0002-8349-7985
Blanca Guadalupe Estrada-Rentería[2]
Estela Lizbeth Muñoz-Andrade[3]

Recibido: 31 de agosto 2018
Aprobado: 10 de enero 2019

Resumen

El aprendizaje de la programación es un tópico difícil para los estudiantes universitarios que inician estudios en carreras afines a las ciencias de la computación. Este aprendizaje exige desarrollar habilidades de resolución de problemas mediante estructuras básicas para diseñar algoritmos y programas. De manera simultánea, el alumno debe aprender la sintaxis de un lenguaje de programación, --un entorno de desarrollo integrado (IDE)-- y desarrollar modelos mentales correctos. La combinación de estos requerimientos frecuentemente conlleva una sobrecarga cognitiva en el estudiante. La teoría de carga cognitiva (TCG) propone mecanismos de aprendizaje para ayudar a disminuir esta sobrecarga. Uno de ellos es el "efecto del problema por completar".

El presente estudio tuvo como objetivo medir uno de los efectos predichos por la TGC. Con base en esta, se diseñó material didáctico que se utilizó en un cuasiexperimento

Abstract

The learning of Programming is a difficult topic for university students who begin studies related to the computer sciences. This learning requires developing problem-solving skills through basic structures to design algorithms and programs. At the same time, students must learn the syntax of a programming language, an integrated development environment (IDE), and develop correct mental models. The combination of these requirements often leads to cognitive overload in the student. Cognitive Load Theory (CLT) proposes learning mechanisms to help reduce this overload. One is the "effect of the problem to be completed."

The objective of this study was to measure one of the effects predicted by CLT. Based on this, teaching materials were designed and used in a controlled quasi-experiment (applied during the second semester of 2017) with two groups of first semester students enrolled in

1 Universidad Autónoma de Aguascalientes, Departamento de Sistemas de Información, carlos.arevalo@edu.uaa.mx
2 Universidad Autónoma de Aguascalientes, Departamento de Sistemas Electrónicos, blanca_g_e@hotmail.com
3 Universidad Autónoma de Aguascalientes, Departamento de Sistemas Electrónicos, elmunoz@correo.uaa.mx

controlado (aplicado durante el segundo semestre de 2017) con dos grupos de estudiantes de primer semestre de Ingeniería en Sistemas Computacionales de la Universidad Autónoma de Aguascalientes (UAA). El grupo experimental (n = 42) utilizó el material didáctico diseñado con la TCC, y el grupo de control (n = 47) utilizó material didáctico tradicional. La prueba de diferencia de medias mostró una diferencia estadísticamente significativa (p = 0.002) entre el rendimiento final de ambos grupos. El estudio concluye que los ejercicios por completar tuvieron un efecto positivo en el aprendizaje de los alumnos del grupo experimental, permitiendo una mejor adquisición de esquemas de programación en la forma de planes de programación. Posteriores réplicas aleatorizadas permitirán corroborar o descartar el efecto encontrado.

Palabras clave

Lenguajes de programación (computadores electrónicos), ingeniería de computación, administración de bases de datos, compresión de datos (computadores).

the Computer Systems Engineering from the Universidad de Aguas Calientes (UAA, given its Spanish acronym). The pilot group (n = 42) used the teaching material designed with CBT, and the control group (n = 47) used traditional teaching material. The mean difference test showed a statistically significant difference (p = 0.002) between the final performance of both groups. The study concludes that the exercises to be completed had a positive effect on the learning process of the students in the experimental group, allowing for a better acquisition of programming schemes in the form of programming plans. Therefore, subsequent random replicates will allow to verify or discard the effect found.

Keywords

Programming languages (electronic computers), computer engineering, database management, data compression (computers).

Introducción

El aprendizaje de la programación es un tópico de difícil dominio para estudiantes universitarios que inician sus estudios en programas educativos afines a las ciencias computacionales. Suelen reportarse cifras de reprobación y deserción en países desarrollados de entre 30 y 50% (Guzdial, 2002; Kinnunen & Malmi, 2006). Históricamente, en la UAA los porcentajes de reprobación de las materias introductorias de programación rondan el 30 y el 40 %.

El problema ha adquirido especial relevancia debido al incremento en el uso de tecnologías de información en dispositivos móviles y en ámbitos de la vida cotidiana, con la consecuente necesidad de contar con cada vez más desarrolladores de *software*.

La dificultad de programar radica en que el aprendiz debe adquirir diversas habilidades de forma acumulativa y jerárquica (Jenkins, 2002; Mow, 2008), las cuales consisten en el desarrollo del pensamiento algorítmico, la solución de problemas usando este tipo de pensamiento, el aprendizaje de conceptos de programación y sus correspondientes modelos mentales, y el dominio de la sintaxis específica de un lenguaje de programación junto con su entorno. En cursos tradicionales, es frecuente que estos tópicos se impartan de forma simultánea, lo que suele provocar una sobrecarga mental en el estudiante novato.

En este contexto, investigadores han propuesto diversos métodos y herramientas para atender la problemática. Por ejemplo, Stripeikait (2017) reporta una herramienta basada en juegos serios y sintaxis simplificada para ayudar en la transición hacia lenguajes de programación formales. También, Tahy & Czirkos (2016) proponen ajustes basados en la enseñanza de "lenguajes de programación textuales" como primer lenguaje de programación. Otra herramienta propuesta consiste en el uso del micromundo (Xinogalos, 2010), para la enseñanza de la programación orientada a objetos. Otros investigadores sugieren ajustes al modelo de enseñanza (Malik & Coldwell-Neilson, 2017), aplicando enfoques de mejora continua y aseguramiento de la calidad.

Una revisión integral de las categorías de herramientas desarrolladas para apoyar el aprendizaje de la programación se puede ver en Kelleher & Pausch (2005).

Todos estos estudios aportan resultados positivos, pero tienen como prerequisito ya sea el uso de herramientas de apoyo diseñadas por los propios investigadores (y por tanto de acceso limitado) o la incorporación de nuevos modelos de enseñanza en el currículo.

Método

En el contexto de las ciencias cognitivas, se han hecho aportaciones importantes sobre la forma en que el cerebro humano procesa la información y su efecto medible en el proceso de enseñanza-aprendizaje. Específicamente, la TCG (J. Sweller, 1988; John Sweller, Ayres, & Kalyuga, 2011) describe que el ser humano cuenta con dos tipos de memorias: la de corto y la de largo plazo. La de corto plazo es volátil y de capacidad limitada, con estudios recientes sugiriendo que podemos almacenar y procesar entre 3 y 5 "elementos significativos" de manera simultánea (Cowan, 2010). La de largo plazo se asume de capacidad ilimitada y permanente. Toda la información no sensorial pasa por la memoria de corto plazo, y aquello que sobrepasa su capacidad provoca una "carga cognitiva" y dificulta el aprendizaje. Diseños instruccionales deficientes suelen provocar tal sobrecarga cognitiva. El aprendizaje y la automatización cognitiva resultan de la adquisición de "esquemas" (Cooper & Sweller, 1987; Rist, 1989, 2004), los cuales son construcciones mentales que permiten procesar múltiples elementos como uno solo, que se almacenan en la memoria de largo plazo y que no demandan recursos excesivos de la de corto plazo. Estos esquemas se adquieren con la práctica y la repetición. El ejemplo clásico del uso de esquemas en el aprendizaje se da en la lectura, cuando el ser humano aprende a reconocer palabras completas en lugar de letras individuales.

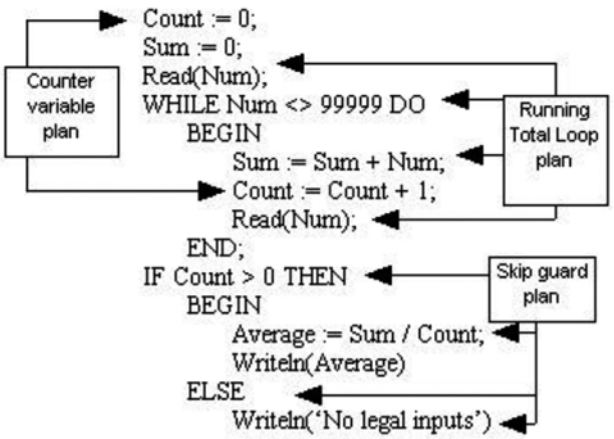
El efecto del ejemplo resuelto y del problema por completar

Los ejemplos resueltos,[4] son soluciones "paso a paso" que reducen la carga cognitiva y permiten la adquisición de esquemas de solución de problemas al plantear problemas equivalentes al estudiante. Se predice que al aplicar esta técnica de enseñanza se evita que el aprendiz utilice estrategias de solución de tipo prueba y error[5] que generan sobrecarga cognitiva. Sin embargo, también se ha observado que, en ciertas circunstancias, el uso de ejemplos resueltos puede derivar en aprendizaje pasivo (Chi, Bassok, Lewis, Reimann, & Glaser, 1989). De tal suerte, una estrategia de enseñanza híbrida consiste en diseñar "problemas por completar", los cuales son ejemplos parcialmente resueltos, en los que el estudiante debe completar algunos pasos clave (Chang, Chiao, & Hsiao, 1996; Hashim & Salam, 2009; van Merrienboer & Krammer, 1990).

Planes de programación

Asociado con los esquemas, en programación existe el concepto de planes (Garner, 2002) los cuales son soluciones estereotipadas para problemas recurrentes (ver figura 1).

Figura 1. Ejemplo de planes de programación.



Tomado de (Garner, 2002)

Por ejemplo, el uso de un "contador" en programación implica la utilización de una variable, una estructura de repetición y una variable tipo acumulador para controlar la cantidad de veces que debe repetirse un ciclo. Con la práctica, todos estos elementos llegan a aprenderse como una sola estrategia, conocida como "contador". Para el presente estudio, se identificaron ocho planes de programación derivados del material didáctico de cursos previos de la materia Introducción a la Programación (ver figura 2), de la carrera de Ingeniería en Sistemas Computacionales de la UAA.

4 El término en inglés es *Worked Example*
5 En inglés, el término utilizado es *Means-Ends Analysis*

Figura 2. Planes de programación, introducción a la programación, UAA

#	Plan	Tipo
1	Plan condicional simple	Selectivo
2	Plan condicional doble	
3	Plan condicional múltiple	
4	Plan ciclo MIENTRAS	Repetición
5	Plan ciclo REPETIR	
6	Plan ciclo PARA	
7	Plan contador / acumulador	Contadores
8	Plan Arreglos / Vectores	Arreglos

En este contexto, se partió de la premisa de que los planes de programación se aprenden como esquemas; y estos pueden adquirirse de forma más eficaz utilizando problemas por completar para reducir la carga cognitiva de la memoria de corto plazo.

Conducción del estudio

La población objetivo del estudio fueron estudiantes de primer semestre de carreras afines a las ciencias computacionales de la UAA, cursando la materia Introducción a la Programación. De tal población, se formaron dos grupos de primer semestre con estudiantes de la carrera de Ingeniería en Sistemas Computacionales de la UAA. Los grupos fueron homogéneos. Esto es, con características similares de conocimiento previo de programación y promedios igualmente similares del nivel educativo anterior. Los grupos no fueron aleatorizados, por lo que el diseño del estudio debe entenderse como cuasiexperimental. Cabe mencionar que el proceso de admisión de la UAA conforma los grupos de primer semestre de las carreras de manera aleatoria. Se controló la variable estilo de enseñanza, siendo la misma docente para ambos grupos. La variable dependiente fue el nivel de aprendizaje de programación básica, medido mediante exámenes parciales estandarizados y calibrados por la academia de computación del Departamento de Sistemas Electrónicos. La variable independiente fue el método de repaso de los ejercicios de programación. Para el grupo de control, el método fue el tradicional, consistente en exposición del maestro, ejercicios de ejemplo y resolución de ejercicios en forma de tareas. El tratamiento del grupo experimental fue el método de repaso por medio de ejercicios por completar (ver figura 3) basados en la TCC, de seis planes de programación aplicados durante cuatro semanas. Se solicitó al grupo experimental realizar estos ejercicios por completar utilizando la herramienta de apoyo a la programación conocida como PseInt[6] (ver figura 4).

Figura 3. Ejemplo de ejercicio por completar

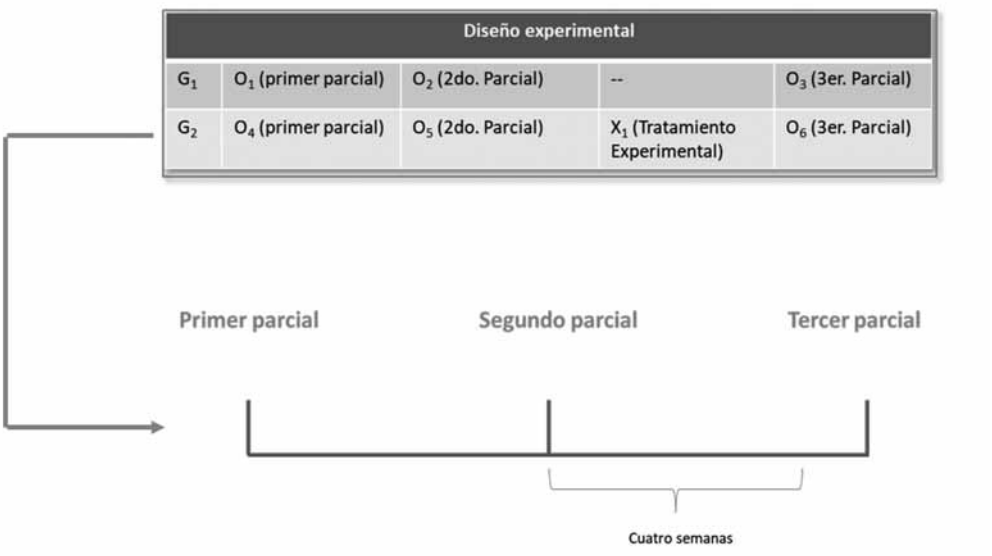
```
1  Algoritmo SI_ENTONCES_SINO
2      Leer NUMERO
3      SI _____________ Entonces
4      |   Imprimir "Numero positivo "
5      _______
6      |   SI _____________ Entonces
7      |   |   Imprimir "Numero negativo"
8      |   Sino
9      |   |   Imprimir "Nulo"
10     |   FinSi
11     FinSi
12 FinAlgoritmo
```

6 Pablo Novara, Copyleft 2003-2017.

Figura 4. Ejemplo de solución de ejercicio por completar en PSeInt

El diseño cuasiexperimental del estudio puede verse en la figura 5. Cabe mencionar que, para una mejor contextualización y un mejor análisis de los resultados, se tomaron en cuenta dos mediciones previas, que fueron los dos primeros exámenes parciales del semestre.

Figura 5. Diseño cuasiexperimental



Resultados

La estadística descriptiva resultante de las calificaciones parciales puede verse en la tabla 1, en la cual se observa que en ambos grupos la media fue decreciendo. En el grupo de control, las medias fueron de 84.54, 67.67 y 52.65, respectivamente. Para el grupo experimental, las medias de cada parcial fueron 78.17, 75.80 y 72.01. El grupo de control inició con una media mayor (84.54), pero los resultados fueron decreciendo en mayor grado hasta ser un valor reprobatorio en el tercer parcial. El grupo experimental inició con una media menor (78.17), pero el decremento fue marginalmente menor y la media del tercer parcial se mantuvo dentro del rango de notas aprobatorias (mayor que 70 puntos).

Tabla 1. Estadística descriptiva. Resultados de exámenes parciales

Grupo	1er examen parcial			2° examen parcial			3er examen parcial		
	Media	Mediana	Moda	Media	Mediana	Moda	Media	Mediana	Moda
Exp (*n* = 42)	78.17	81.55	93.00	75.80	75.87	90.00	72.01	75.37	100
Control (*n* = 47)	84.54	91.00	100.00	67.67	75.00	93.5	52.65	51.95	.00

Comparación de medias

Los resultados de las pruebas t de comparación de medias entre los resultados de los tres exámenes parciales de los dos grupos, tanto en la forma de comparación de muestras dependientes (es decir, los resultados de un mismo participante dentro del grupo, en cada parcial), así como de muestras independientes entre los grupos experimentales y de control, se muestran en las tablas 2, 3 y 4.

Tabla 2. Prueba t, primer y segundo examen parcial, grupos experimental y de control

Prueba de muestras relacionadas

		Diferencias relacionadas					t	gl	Sig. (bilateral)
		Media	Desviación típ.	Error típ. de la media	95% Intervalo de confianza para la diferencia				
					Inferior	Superior			
Par 1	PrimerParcialExp - SegundoParcialExp	2.36190	10.41775	1.60749	-.88450	5.60831	1.469	41	.149
Par 2	PrimerParcialCtrl - SegundoParcialCtrl	16.87021	11.64067	1.69797	13.45238	20.28804	9.936	46	.000

Tabla 3. Prueba t, segundo y tercer examen parcial, grupos experimental y de control

Prueba de muestras relacionadas

		Diferencias relacionadas					t	gl	Sig. (bilateral)
		Media	Desviación típ.	Error típ. de la media	95% Intervalo de confianza para la diferencia				
					Inferior	Superior			
Par 1	SegundoParcialExp - FinalExp	3.79286	11.93130	1.84104	.07480	7.51091	2.060	41	.046
Par 2	SegundoParcialCtrl - FinalCtrl	15.01489	17.59314	2.56622	9.84936	20.18043	5.851	46	.000

Tabla 4. Prueba t, tercer examen parcial, grupos experimental y de control

Prueba de muestras independientes

		Prueba de Levene para la igualdad de varianzas		Prueba T para la igualdad de medias						
									95% Intervalo de confianza para la diferencia	
		F	Sig.	t	gl	Sig. (bilateral)	Diferencia de medias	Error típ. de la diferencia	Inferior	Superior
Calific	Se han asumido varianzas iguales	18.237	.000	3.203	87	.002	19.35922	6.04482	7.34448	31.37396
	No se han asumido varianzas iguales			3.281	78.748	.002	19.35922	5.89970	7.61559	31.10285

Tales pruebas indican que en el grupo de control hubo diferencia significativa en el comportamiento de los resultados de las tres calificaciones parciales. En otras palabras, en cada distribución, las calificaciones se comportaron de manera diferente. Para el grupo experimental, las comparaciones entre las calificaciones del primer y segundo examen parcial intragrupo muestran que no hubo diferencia significativa. En la comparación de medias entre el segundo y tercer parcial si se observa una diferencia estadística significativa (p = 0.046).

La comparación de medias de muestras independientes entre los resultados del tercer parcial (es decir, posterior al tratamiento) de los grupos de control y experimental arroja un valor p=0.02, lo cual indica también una diferencia estadísticamente significativa entre ambos grupos.

Discusión

Las pruebas de comparación de medias dependientes (es decir, las pruebas pareadas) del primer y segundo parcial no mostraron diferencias. En otras palabras, no hubo cambios significativos (p = 0.149) en el rendimiento académico entre estas dos observaciones. Por otro lado, la prueba pareada de comparación de medias entre el segundo y tercer parcial de este mismo grupo experimental sí muestra un cambio estadístico significativo (p = 0.046), lo que proporciona indicios de un efecto debido al tratamiento basado en problemas por completar, ya que este tuvo lugar entre estas dos observaciones. Tal efecto también es visible con la prueba de comparación de medias de muestras independientes del tercer parcial entre los grupos experimental y de control (p = 0.002). Como confirmación adicional, las medias de ambos grupos tuvieron una diferencia de casi un 20 % (19.36 puntos).

Por lo anterior, puede asumirse que el uso de ejercicios basados en los planes de programación identificados, y estos a su vez basados en problemas por completar sugeridos por la TCC, tuvo un efecto positivo en el aprendizaje de los alumnos. El efecto predicho por esta teoría, en el sentido de que la carga cognitiva extrínseca sobre la memoria de corto plazo, al utilizar problemas por completar, es menor que el uso de estrategias de solución de tipo prueba y error, se corrobora mediante los resultados obtenidos. Por otro lado, debe mencionarse que no es posible aún hacer una generalización de los resultados, dado que el diseño experimental no contó con aleatorización, pero consideramos que de manera preliminar estos son fiables, debido a la homogeneidad de los grupos conformados en el estudio.

Los resultados obtenidos son compatibles con otros estudios relacionados con la teoría de la autoeficacia, la motivación y el trabajo colaborativo (Arévalo, Andrade, & Reynoso, 2018; Bandura, 1982; Pintrich & Zusho, 2007), en los que el incremento gradual en la dificultad de las actividades fomenta precisamente la motivación y la autoeficacia en el estudiante, variables que a su vez tienen un efecto positivo en el rendimiento académico.

Estudios futuros en esta línea de investigación requerirán de réplicas con diseños aleatorizados y de la identificación de otros planes de programación derivados del material didáctico de otros docentes participantes. También creemos que es factible extender el alcance del estudio mediante el diseño de problemas por completar en contenidos de programación intermedia y avanzada, de programación orientada a objetos o incluso en algunos dominios de matemáticas básicas.

Referencias

Arévalo, C., Andrade, E.L.M., & Reynoso, J.M.G. (2018). "El efecto de la autoeficacia y el trabajo colaborativo en estudiantes novatos de programación". *Investigación y Ciencia: de la Universidad Autónoma de Aguascalientes*. (74), 73-80.

Bandura, A. (1982). *"Self-Efficacy mechanism in human agency"*. American Psychologist. 37(2), 122-147.

Chang, K.-E., Chiao, B.-C., & Hsiao, R.-S. (1996). *"A programming learning system for beginners — A completion strategy approach". Intelligent Tutoring Systems*. 623-631. https://doi.org/10.1007/3-540-61327-7_162

Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P., & Glaser, R. (1989). *"Self-explanations: How students study and use examples in learning to solve problems"*. Cognitive Science. 13(2), 145-182. https://doi.org/10.1016/0364-0213(89)90002-5

Cooper, G., & Sweller, J. (1987). *"The effects of schema acquisition and rule automation on mathematical problem-solving transfer". Educational Psychology Review.* 79, 347-362.

Cowan, N. (2010). "*The magical mystery four: how is working memory capacity limited, and why?". Current Directions in Psychological Science*. 19(1), 51-57. https://doi.org/10.1177/0963721409359277

Garner, S. (2002). *"The learning of plans in programming: a program completion approach". International Conference on Computers in Education*. 2, 1053-1057. https://doi.org/10.1109/CIE.2002.1186149

Guzdial, M. S. (2002). *"Teaching the Nintendo generation how to program". Communications of the ACM,* 45(4), 17-21.

Hashim, N., & Salam, S. (2009). *"Integration of visualization techniques and completion strategy to improve learning in computer programming". International Conference of Soft Computing and Pattern Recognition*. 665-669. https://doi.org/10.1109/SoCPaR.2009.131

Jenkins, T. (2002). *On the difficulty of learning to program.* Loughborough University: LTSN Centre of information and computer sciences.

Kelleher, C.P., & Pausch, R. (2005). *"Lowering the barriers to programming: a survey of programming environments and languages for novice programmers"*. ACM Computing surveys (CSUR), 37(2), 83-137.

Kinnunen, P., & Malmi, L. (2006). *"Why students drop out CS1 course?". Proceedings of the Second International Workshop on Computing Education Research*. 97-108. https://doi.org/10.1145/1151588.1151604

Malik, S.I., & Coldwell-Neilson, J. (2017). *"A model for teaching an introductory programming course using ADRI". Education and Information Technologies*, 22(3), 1089-1120. https://doi.org/10.1007/s10639-016-9474-0

Mow, I. T. C. (2008). Issues and difficulties in teaching novice computer programming. *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education*, 199-204.

Pintrich, P.R., & Zusho, A. (2007). *"Student motivation and self-regulated learning in the college classroom". The Scholarship of Teaching and Learning in Higher Education: An Evidence-Based Perspective.* 731-810. https://doi.org/10.1007/1-4020-5742-3_16

Rist, R.S. (1989). "*Schema Creation in Programming". Cognitive Science*. 13, 389-414.

Rist, R.S. (2004). *"Learning to program: schema creation, application and evaluation". Computer Science Education and Research*. 175-197.

Stripeikait, I. (2017). *"'Skipping the Baby Steps': The importance of teaching practical rogramming before programming theory". Serious Games.* 319-330. https://doi.org/10.1007/978-3-319-70111-0_30

Sweller, J. (1988). *"Cognitive Load During Problem Solving: Effects on Learning". Cognitive Science*. 12, 257-285.

Sweller, J., Ayres, P., & Kalyuga, S. (2011). *Cognitive Load Theory*. New York, NY: Springer.

Tahy, Z.S., & Czirkos, Z. (2016). *"'Why can't I learn programming?'. The learning and teaching environment of programming". Informatics in Schools: Improvement of Informatics Knowledge and Perception*. 199–204. https://doi.org/10.1007/978-3-319-46747-4_17

Van Merrienboer, J., & Krammer, H. (1990). *"The 'completion strategy' in programming instruction: Theoretical and empirical support". Research on Instruction*. 45-61.

Xinogalos, S. (2010). *"An interactive learning environment for teaching the imperative and object-oriented programming techniques in various learning contexts". Knowledge Management, Information Systems, E-Learning, and Sustainability Research*. 512-520. https://doi.org/10.1007/978-3-642-16318-0_66