Programación concurrente: una aproximación a las dificultades de su aprendizaje

Alejandro Miños[1], Juan Sales[2]

[1] Instituto Normal de Enseñanza Técnica, Montevideo, Uruguay
[2] Instituto Normal de Enseñanza Técnica, Montevideo, Uruguay

{alejandromifa , juanp.sales.gonzalez}@gmail.com

Resumen

El estudio de las técnicas de programación es fundamental en las carreras de grado en el área de la Computación, pues estas son la base de los programas informáticos. Al mismo tiempo, los sistemas operativos modernos solo podrán ser comprendidos si a su vez pasa lo mismo con la programación concurrente, técnica que permite la ejecución coordinada de varios procesos.

El presente trabajo ha tomado como población alumnos de formación docente, en la especialidad Informática, analizando las principales dificultades observadas en relación a esta técnica de programación.

Tomando como indicadores los resultados observados en los parciales y trabajos obligatorios, hemos obtenido dos conclusiones principales: el uso de un lenguaje de programación conocido por los alumnos explica parte del éxito obtenido; por otra parte, el trabajo con ejemplos teóricos, considerados paradigmáticos, es útil, y puede aumentar el entendimiento en la materia, aunque no es suficiente para lograr la construcción de programas viables y entender la programación concurrente como técnica.

Palabras clave: programación concurrente, sistemas operativos, didáctica de la informática

Abstract

The study of programming techniques is essential in undergraduate courses pf computer science, since programs are the building block for all software. At the same time, modern operating systems can only be understood with a previous understanding of concurrent programing, that is the way to have several programs running in parallel.

This investigation worked with a population of college students of computer science. analizing the main difficulties encountered in relation to this programming technique.

Based on the results observed in tests and assignements, we reached two main conclusions: 1) The use of a programming language known by the students is part of the success. 2) Work with theoretical examples, some of which are considered paradigmatic, is useful, and can help in the understanding on the subjet, but it is not enough to achieve the construction of viable programs and understanding concurrent programming as a technique.

Keywords: Concurrent programming, operating systems, computer didactics.

1. Introducción

No es posible pensar la Computación sin el estudio de las técnicas de programación. La comunicación entre computadoras solo es posible en tanto y cuanto esto se programe. La robótica, más allá de los aspectos electrónicos inherentes a ella, solo será posible si existen algoritmos que determinen el funcionamiento del robot. Los sistemas operativos, la administración de los recursos de software y hardware de una computadora, deberán ser programados a fin de poder administrar la memoria, las unidades de almacenamiento secundario o la ejecución de varios programas a la vez. De este modo, la programación es estructurante de los currículos de Informática, pues en torno de ella giran buena parte de las áreas de las Ciencias de la Computación (a no ser los fundamentos teóricos).

Programar es realizar algoritmos informáticos, los cuales consisten en un conjunto de instrucciones que pueden ser ejecutadas en un computador, que al hacerlo crea un proceso, el cual es pues una instancia de un programa en ejecución tal cual establece Eden [1]. Los procesos, al estar en ejecución, utilizan recursos de hardware y del Sistema Operativo, por

ejemplo: Memoria Principal, dispositivos de Entada/Salida, tiempo de CPU.

Se entiende por concurrencia la ejecución de varios programas a la vez, lo que lleva a que los procesos asociados a estos compartan y compitan por los recursos de hardware y software, teniendo a su vez cada proceso un conjunto de recursos propios, con los cuales trabaja y le permiten un correcto funcionamiento [2]. Dado que los procesos pueden competir entre ellos por un mismo recurso, como ser espacios de memoria o almacenamiento secundario, y para evitar las interferencias en el acceso a ellos, los distintos procesos deben coordinar el uso de dicho recurso, de forma ordenada y de tal modo que la apropiación del mismo por parte de uno de ellos no genere inconsistencias en el trabajo de otros procesos. En los actuales sistemas operativos multitarea, la escritura de los programas debe realizarse considerando que al ejecutarse, el proceso generado comparte y compite por recursos con otros procesos.

La programación concurrente en ocasiones es tratada como una asignatura independiente de otras y en otra es trabajada transversalmente en los programas de varias asignaturas del área Informática: Sistemas Operativos, Bases de Datos o Sistemas Distribuidos entre otros. Esta técnica de programación es esencialmente distinta a otras formas de programar, donde se programa como si el proceso fuera el único que se ejecuta en la computadora, siendo necesario el aprendizaje de nuevo paradigma de programación por parte del estudiante. El estudio de la programación concurrente permite:

1. Ejemplificar y fortalecer la relación que existe entre la programación y el resto de las asignaturas de las carreras de grado del área de Computación.
2. Entender la programación, en sus distintas variantes, como un conjunto de técnicas estructurantes de los planes de las carreras de Computación, los cuales permiten identificar el componente experimental de la ciencia [3].
3. Conocer y aplicar una técnica de programación de amplio uso en la actualidad.

Tanto la enseñanza como el aprendizaje de la programación concurrente, se presentan como difíciles, sobre todo cuando forma parte de otras asignaturas y no es trabajada como una asignatura independiente. En efecto, el resolver problemas concurrentes implica que el alumno se apropie de métodos de trabajo con los cuales no solo no está familiarizado, sino que no están relacionados con otros ya vistos.

Entender las dificultades con las cuales se encuentran los alumnos, en el aprendizaje de la programación concurrente, contribuirá a mejorar las estrategias de enseñanza relacionadas con esta técnica de programación.

2. Marco teórico

2.1. Estudios previos

La revisión bibliográfica ha relevado algunas investigaciones relacionadas con la programación paralela, caso particular de la programación concurrente, aunque no centradas en lo didáctico. Recordemos que la programación en paralelo asume la existencia de varios núcleos en el procesador, o varios procesadores, en tanto que la concurrencia estudia la existencia de varios procesos interactuando.

El artículo de nombre "¿concurrencia y Paralelismo en el primer curso de Algorítmica?" de Armando De Giusti y Fernando Frati [4] del Instituto de Investigación en Informática de la Facultad de Informática de la Universidad Nacional de La Plata, analiza el abordaje temprano de la concurrencia en los planes de estudio. Los autores sostienen que la aparición de los procesadores multi núcleos han impulsado "el agotamiento definitivo del modelo Von Neumann". El avance tecnológico hace que la programación clásica, serial, "será sólo un "caso degenerado" poco aplicable, ya que conduciría a tener N-1 núcleos del procesador ociosos". Atendiendo al hecho que el estudio de la concurrencia se realizar relativamente tarde en los cursos de computación, y considerando la dificultad de su abordaje, los autores proponen incluirlo tempranamente en los cursos de Computación. Los autores analizan el uso del entorno de trabajo LMRE, destacando el hecho que el mismo permite visualizar el uso de conceptos concurrentes, fomentando a su vez el proceso de abstracción necesario para ello.

En el artículo "Sobre el papel de la programación paralela en los nuevos planes de estudio de informática", de F. Almeida, D. Giménez, J. Mantas y A. Vidal [5], analiza la incorporación de esta técnica en los planes de estudio de las carreras informáticas. Los autores analizar tanto el caso de la programación concurrente como de la paralela. El trabajo con conceptos de concurrencia y paralelismo, y su aplicación, se da en distintas asignaturas y formas: sistemas operativos, arquitectura de computadores, aplicaciones y obligatorios de asignaturas y en menor medida en asignaturas que tienen por objetivo la programación paralela. Los autores advierten que el abordaje de estos contenidos desde distintas enfoques y asignaturas puede dar lugar a la conformación de titulares, siendo necesario "unificar posturas en cuanto a los conocimientos de los algoritmos paralelos que deberían incluirse en los próximos planes de estudio". El trabajo pretende fomentar la discusión sobre la temática, con alguna asignatura específica, considerando que un profesional del área debería poder manejar técnicas de programación paralela.

2.2. Concurrencia y planes de estudio

Como ya dijimos, la programación concurrente, es

abordada desde dos lugares distintos: como parte de otra asignatura como ser Sistemas Operativos, o como una asignatura independiente en sí misma. Ambos enfoques tienen características propias y diferenciadas:

1. Inclusión en otra asignatura: en este caso la programación concurrente será un medio para trabajar otros contenidos u objetivos. Un ejemplo de esto es el estudio de los sistemas multitarea, lo cual solo puede ser abordado desde la concurrencia. Este enfoque trabaja colateralmente el estudio de la programación concurrente, lo que puede quitarle tiempo de estudio, incrementando la dificultad de aprendizaje si el abordaje y los objetivos no han sido cuidadosamente estudiados.
2. Asignatura independiente: este enfoque considera que los contenidos a considerar son lo suficientemente importantes como para darles un espacio propio. Desde esta perspectiva si bien algunos aspectos de la programación concurrente podrían ser abordados tempranamente en otras asignaturas, se debe generar un espacio propio y diferenciado. Es así que estaríamos trabajando con un nuevo paradigma de programación, generándose una dificultad significativa en el proceso de aprendizaje del paradigma, lo que se ha denominado como el "problema del desplazamiento" [6].

Tres son las formas de programar concurrentemente: semáforos, mensajes y monitores. Mientras que el uso de monitores, como en el caso de la sincronización de Java [7], hace que el programador se desentienda fuertemente de la administración de la concurrencia, el trabajo con semáforos o mensajes exige un mayor detenimiento en la administración de los procesos concurrentes. Con las dos últimas herramientas, el programador debe manejar la lógica de la concurrencia, a la vez que la sintaxis de los métodos o funciones del Sistema Operativo que permiten hacerlo. Si bien las tres herramientas de codificación tienen asociada una sintaxis específica, en el caso de los semáforos y mensajes esto se complejiza, pues para operar con ellos se deben definir previamente espacios de memoria compartida, variables, colas de mensajes o registros especiales de funcionamiento en los mensajes. Por tanto, la opción más sencilla sería el uso de los monitores de Java, elemento que se ve dificultado por el aprendizaje de un lenguaje nuevo en caso de no haberlo trabajado.

En nuestro caso elegimos el uso de semáforos por cinco motivos:

1. Implica usar un lenguaje que ya es conocido por los estudiantes, evitando la sobrecarga cognitiva. En efecto, C provee el manejo de semáforos, por lo que el estudiante no necesita aprender un nuevo lenguaje de programación.
2. El uso de semáforos se asemeja al de una llave, en tanto que los mensajes (la otra opción a usar con el lenguaje C) implica el enviar y recibir estructurados, emulando el envío de una correspondencia, a priori menos intuitiva que una llave.
3. La sintaxis en el uso de semáforos es más sencilla que en el caso de mensajes.
4. Si bien es necesario definir las variables de tipo semáforo, espacios de memoria compartidos y asignar permisos, lo anterior puede ser abordado de forma axiomática, sin una gran dificultad para los alumnos.
5. El objetivo del presente trabajo es aproximarnos a las dificultades asociadas a la programación concurrente, y no un curso de concurrencia, por lo que no es relevante el trabajo con varias herramientas que permitan este tipo de programación.

2.3. Aprendiendo a programar

Múltiples artículos versan sobre la enseñanza y el aprendizaje de la programación, estando centrados en la enseñanza universitaria principalmente. Algunos elementos a considerar, y que se presentan en reiteradamente:

1. Necesidad de entender el problema a trabajar, limitándolo convenientemente.

   La resolución de un problema implica que el estudiante identifique dicho problema, a fin de poder resolverlo. Desde esta perspectiva el proceso de Análisis y Diseño de una aplicación no puede ser separada del desarrollo, al menos sin incurrir en artificialidades que atentan contra el producto y el aprendizaje. Si bien analizar y diseñar un sistema sería parte de asignaturas del área de la Ingeniería de Software, se considera que no es posible programar sin manejar al menos algunos conceptos y técnicas básicas de este tipo. Desde las asignaturas del área programación, se trabajarían algunas destrezas que permitan delimitar el problema, para luego identificar una solución óptima. La correcta identificación del problema a programar está asociada a reducir la frecuencia y tipos de errores en los cuales incurre el estudiante a la hora de programar [8]. Al mismo tiempo, será necesario diferenciar el análisis del problema de la solución de "problemas tipo", o del uso de técnicas de modelado (como ser pseudo código), pues en ocasiones "los estudiantes parecen haber aprendido de memoria" [9].
2. Presentar actividades que resulten motivantes, relacionadas con el trabajo cotidiano, que resulten significativas para los estudiantes

[10].

Si bien el uso de ejemplos matemáticos (promediar un conjunto de valores numéricos, hallar el máximo o calcular un promedio) puede facilitar la realización de actividades por parte del profesor, estas no necesariamente resultan interesantes para los estudiantes. Como ejemplo citamos el hechos que técnicamente no es necesario operar con variables de tipo Entero para ordenar datos, pudiendo ser sustituidas por fechas o nombres de personas, según las características del lenguaje usado. Cognitivamente las operaciones relevantes serán las iteraciones, asignaciones o comparaciones, entre otras, por lo que tampoco se justificaría el uso de ejemplos matemáticos.

3. En la misma línea de lo expresado anteriormente, debemos considerar la motivación por el logro de los estudiantes.

La creación de actividades que permitan que el alumno realice ejercicios de metacognición, y que permitan "buscar una explicación al resultado" obtenido, para así poder "atribuir una causa al éxito o al fracaso" [11], aumentarán la motivación de los estudiantes. Es así que no solo será relevante que el profesor aborde ciertos contenidos programáticos, sino que además los mismos deben permitir que el estudiante los sienta como propios, motivándolo así en su proceso de aprendizaje. Del mismo modo, si bien aprender sintaxis no es aprender a programar, no debemos olvidar que si el programa no es implementado en una computadora no podrá ser testeado por los estudiantes, a no ser de forma teórica, lo que puede afectar la motivación de los estudiantes y la auto evaluación.

4. Considerar los tiempos necesarios para aprender un nuevo paradigma de programación habiendo aprendido previamente otro.

Este pasaje de un paradigma a otro, como ser el pasaje de la programación estructurada – imperativa a la orientada a objetos, implica que el estudiante se apropie de "formas de pensar" que le son nuevas, lo que se ha dado en llamar el "problema del desplazamiento" [12]. La evidencia empírica muestra que para un programador experto, el desplazamiento de la programación estructurada a la orientada a objetos implica un aprendizajede entre seis y dieciocho meses aproximadamente [6]. En el caso de la programación concurrente los problemas indicados se mantienen en su totalidad no solo por ser una técnica de programación, sino por la metodología usada habitualmente y referenciada bibliográficamente. Ya sea si la programación concurrente es tratada como una asignatura independiente o incluida en cursos como Sistemas Operativos, lo cierto es que el estudiante ya ha transitado un por un camino en el cual lo común es la programación no concurrente. El trabajo con sistemas que se ejecutan en serie es lo estándar en los cursos introductorios de programación, o en aquellos que trabajan con tipos e datos abstractos u orientados a objetos. Esta secuenciación de cursos que no abordan la programación concurrente, hace que el estudiante se encuentre en un escenario donde debe aprender a evaluar elementos que hasta el momento eran considerados como datos que no generaban una dificultad significativa. De aquí en adelante, el estudiante deberá identificar y operar adecuadamente los recursos por los cuales compiten los procesos, algunos de los cuales están explícitos y otros no. El hecho de codificar soluciones, programar, implicará que el estudiante nuevamente se enfrente al problema de aprender un lenguaje, o al menos manejar nuevas funciones o procedimientos que permitan programar. Lo anterior puede dar lugar a la sobrecarga cognitiva, sobre todo en aquellas asignaturas donde la concurrencia es parte de un contenido más general. Ha aquí donde el docente deben hacer énfasis en las estrategias metodológicas, por ejemplo decidiendo si usará un nuevo lenguaje (desconocido hasta el momento por los alumnos), o trabajará con uno que los estudiantes ya conozcan y dominen.

En relación a los ejemplos usados, estos son recurrentes en el sentido de hacer referencia al problema de los consumidores – productores, lectores – escritores o filósofos entre otros. Si bien consideramos que los mismo resultan muy ilustrativos sobre lo que es y cómo se trabaja la concurrencia, no menos cierto es que pueden parecer alejados de la realidad computacional del estudiante. En efecto, si bien los problemas anteriores modelan razonablemente bien un conjunto de realidades, no menos cierto es que su aplicación puede no resultar trivial para los estudiantes. De este modo, y partiendo de la base que sería una situación extrema, se corre el riesgo que la transposición realizada por docente, no sea significativa para el alumno, convirtiéndose en un mero análisis teórico sin el aprendizaje del nuevo paradigma.

3. Diseño de investigación

3.1. Aspectos metodológicos

Como hemos visto, la cantidad de investigaciones previas relacionadas con las dificultades en el aprendizaje de la programación concurrente es escasa o

nula, siendo el abordaje principalmente curricular. De este modo, realizamos un trabajo principalmente exploratorio y correlacional, en el cual buscamos identificar las principales dificultades relacionadas con la programación concurrente.

La población estudiada fue de treinta y cuatro alumnos, pues solamente se consideraron aquellos estudiantes que finalizaron el curso de Organización del Computador y Sistemas Operativos de segundo año del Profesorado de Informática en el Instituto Normal de Enseñanza Técnica, divididos en cuatro grupos.

Se realizó una primera evaluación teórica, y un posterior obligatorio práctico, donde fueron evaluados los siguientes elementos de la solución dada:

1. Construcción del obligatorio en lenguaje C, resolviendo el problema planteado.
2. Identificación de regiones críticas y solución general del problema.

3.2. Enfoque áulico

En relación al lenguaje de programación, hemos optado por trabajar en C, pues los estudiantes han programado en primer año, y continúan usándolo en segundo año. Al mismo tiempo, la sintaxis de este lenguaje en lo relacionado con la programación concurrente es simple, no siendo necesario realizar ningún pasaje de parámetros ni trabajar con estructuras de datos. Como sistema operativo se utilizo Linux, un clon de Unix, debido a que las primitivas necesarias para la concurrencia son simples y están bien documentadas.

Desde el punto de vista teórico primeramente se analizó el problema de los filósofos, propuesto por Edger Dijkstra [13] y re-formulado por Tony Hoare [14] en su enunciado actual, más didáctico. Otros ejemplos usados fueron el de Bernardo y Blanca (dos vecinos que comparten un patio al que no pueden acceder al mismo tiempo sus perros), el del productor – consumidor y colateralmente los lectores escritores. Sin hacer mucho énfasis en los conceptos fuertes de concurrencia (regiones críticas, bloqueos e inanición) pudimos observar que los estudiantes identificaron los recursos a compartir, las secciones críticas, las condiciones de inanición y la necesidad que dos filósofos que ocupen lugares consecutivos no pueden comer a la vez.

A fin de aproximarnos a una solución desde la programación concurrente, trabajamos el concepto de semáforo como el de una llave, la cual permite el acceso a la sección crítica, buscando que los estudiantes identifiquen el proceso:

Tomar llave

Operar en el recurso

Dejar llave.

Mediante aproximaciones sucesivas pudimos presentar la solución básica general de la forma:

P(Semáforo)

Sección Crítica

V(Semáforo)

Como dijimos la codificación fue realizada en lenguaje C, trabajando únicamente con semáforos binarios. En relación a la definición del semáforo, sus características y la memoria compartida, el trabajo se presentó de forma axiomática; si bien se explicó su funcionamiento, no se hizo énfasis en las posibles variantes de los mismos, trabajándose como cajas negras.

4. Resultados obtenidos

Luego de haber trabajado los conceptos y procedimientos básicos de la programación concurrente se procedió a ejemplificar algunos casos. Posteriormente propusimos un trabajo obligatorio y práctico a los estudiantes, a fin de aplicar los contenidos vistos. Una vez presentada la actividad se tutoreó el trabajo de los estudiantes, sobre todo en lo relacionado con la programación del sistema, elemento que a priori se consideró como una debilidad. Se prestó especial atención a la creación de los distintos procesos.

Corresponde indicar que el problema fue una variante de los lectores – escritores, sin prioridad para ninguno de ellos, en los cuales se trabajaba con memoria estática y sin tipos estructurados.

1. El 30% de los alumnos presentaron problemas graves a la hora de codificar la solución propuesta, logrando hacerlo en pseudo código o con errores muy graves desde el punto de vista programación. Cabe considerar que el sistema de previaturas implica que un alumno puede cursar la asignatura Organización del Computador y Sistemas Operativos sin necesidad de haber cursado / aprobado otras materias del profesorado.
2. Tanto en el manejo de las secciones críticas como en la construcción de procesos, los niveles de correctitud propuestos por los estudiantes son similares, en el entorno del 50% para ambas categorías, consideradas por separado. No solo consideramos una solución correcta aquella que efectivamente lo es, así como las que presentan errores menores, pero que muestran buenas bases teóricas en relación a la concurrencia.
3. Los alumnos que han operado correctamente con las secciones críticas o con los procesos, siendo una de las dos aceptable, nos encontramos que el porcentaje se eleva al 60%. Este grupo de alumnos, estaría en una categoría la cual está formada por aquellos

estudiantes que muestran una comprensión aceptable del problema en general.

4. Si consideramos los estudiantes que han operado de forma aceptable, tanto con las secciones críticas como en la construcción de procesos, podemos ver que el porcentaje es del 38%. Estos alumnos, no solo cumplen los dos criterios indicados en el punto anterior, sino que además lo hacen a la vez, siendo por tanto el grupo que menos dificultades presenta para la construcción de una solución viable. Solo un estudiante, dentro de este grupo, no obtuvo calificación aceptable.
5. Un tercer grupo de alumno lo forman los alumnos que han operado sobre secciones críticas, creación de procesos y además han programado una solución aceptable (con distintos niveles de correctitud). Estos alumnos son el 35% del total, no siendo significativa la diferencia con los del grupo anterior, y muestran un buen trabajo en relación a la programación concurrente.
6. Si consideramos la primera evaluación parcial realizada, de tipo teórica, se observa que no existe correlación entre la resolución de los problemas desde el punto de vista teórico y su posterior codificación.

En síntesis:

Existe un alto porcentaje de estudiantes que presentan dificultades en el manejo de las secciones críticas y los procesos, tanto de forma separada como conjunta.

Las calificaciones de suficiencia están relacionadas con el manejo de los conceptos mencionados anteriormente y el manejo adecuado del lenguaje C. Si bien estos aspectos deben ser relativizados por la construcción del sistema de evaluación en sí mismos, no es menos cierto que estos aspectos son los fundamentales a la hora de programar concurrentemente.

Aquellos alumnos que muestran un manejo aceptable del lenguaje C parecen estar en mejores condiciones para programar concurrentemente, en tanto que aquellos que tienen dificultades con el lenguaje presentan una dificultad mayor al respecto.

5. Conclusiones

Dos grupos de conclusiones se desprenden del presente trabajo: las primeras relacionadas con el proceso de transposición didáctica, y las segundas relacionadas con el aprendizaje del paradigma propiamente dicho.

1. El planteamiento de ejemplos típicos, o paradigmáticos en relación a la programación concurrente, se muestra como especialmente adecuado para la presentación del contenido. En general, este tipo de ejemplos permiten identificar procesos y recursos de modo que acercan al estudiante al problema en cuestión. Sin embargo, este tipo de abordaje de los contenidos tiene sus limitaciones, pues se observa que los estudiantes que presentan buenos desempeños en esta modalidad de trabajo no necesariamente lo hacen cuando deben programar soluciones.

   Es así que el trabajo teórico no sería un buen indicador del conocimiento que tienen los alumnos de la programación concurrente, siendo necesario separar esta metodología de trabajo que aquellas que potencian los aprendizajes de la programación concurrente.
2. Desde la programación se observa una mayor dificultad en el manejo de los procesos que en el caso de las secciones críticas. En el entorno del 50% de los estudiantes han mostrado dificultades al respecto.

   Si bien es cierto que el trabajo obligatorio se reducía a dos procesos, con un solo semáforo, y además binario, debemos tener en cuenta que las actividades prácticas fueron también limitadas y no existieron actividades de repaso de programación estructurada, técnica que se consideraba manejada.

   De este modo sería interesante presentar actividades que simplifiquen el trabajo con la creación de procesos y las secciones críticas, en particular de forma separada e independiente una de otras, para posteriormente ensamblarlas en una única actividad (a diferencia de lo que se hizo en el trabajo obligatorio presentado)
3. Reafirmando lo anterior correspondería analizar formas de trabajo que surjan desde las actividades prácticas y no desde los ejemplos teóricos, pues estos presentan las limitaciones indicadas en el numeral 1.
4. Por último, teniendo en cuenta el uso del lenguaje de programación, y las dificultades propias de la codificación de soluciones, no parece adecuado la incorporación de uno nuevo para le enseñanza de la programación concurrente, pues se observa que éste constituye en sí mismo un obstáculo importante para los estudiantes.

No obstante lo anterior, existen aún algunos elementos sin responder que podrían ser abordados a posteriori, como ser: ¿qué incidencia tienen el uso del lenguaje en el aprendizaje de la programación concurrente?, ¿qué tipos de errores existen en la codificación de las soluciones?, ¿cuál sería la secuencia de actividades que permiten maximizar los aprendizaje de los alumnos?, ¿cuáles son las representaciones mentales que los estudiantes hacen de la programación concurrente?

Referencias

[1] A. Eden, Three Paradigms of computer Science". Minds & Machines, Vol. 17 Issue 2, July 2007 pp. 135 – 167.

[2] A. Tanenbaum, Sistemas operativos modernos. Pretince Hall Hispanoamérica S.A, México, 1992.

[3] P. Denning, Is computer science science?. Communications of the ACM, Vol. 48 Issue 4, April 2005 pp, 27 – 31.

[4] A. De Giusti, A. Frati, ¿concurrencia y Paralelismo en el primer curso de Algorítmica?. V Congreso de Tecnología en Educación y Educación en Tecnología (2010), El Calafate.

[5] F. Almeida, D. Giménez, J. Mantas, A. Vidal, Sobre el papel de la programación paralela en los nuevos planes de estudios de informática. Recuperado el 10 de marzo de 2014 de: http://lsi.ugr.es/jmantas/papers/jenui2009.pdf

[6] D. Gayo, A. Cernuda del Río, J. Cueva, M. Díaz, M. Almudena, J. Redondo, Reflexiones y experiencias sobre la enseñanza de POO como único paradigma, (2003). Recuperado el 20 de abril de 2014 de: di002.edv.uniovi.es/~dani/publications/jenui03.pdf

[7] P. Deitel, & H. Deitel,: Java™ Como Programar. PEARSON Educación, México, 2007.

[8] A. Ebrahimi, Novice programmer errors: language constructs and plan composition,(1994). Recuperado el 20 de febrero de 2014 de: http://www.sci.brooklyn.cuny.edu/~kopec/research/sdarticle1.pdf.

[9] I. Pérez, A. Fuentes, S. Moreno, Estudio de la problemática presente en el diseño de algoritmos por computadora?. III Congreso Universitario de Tecnologías de Información y Comunicación, (2008), Tlahuelilpan.

[10] J. Villalobos, Proyecto cupi2 – una solución integral al problema de enseñar y aprender a programar, (2009) . Recuperado el 10 de marzo de 2014 de: http://www.colombiaaprende.edu.co/html/mediateca/1607/articles-205832_recurso_1.pdf

[11] M. Míguez, ¿Cómo motivar para aprender? En Didáctica práctica para enseñanza media y superior (Fiore, E., Lymonié, J, Ed.), (2007), pp. 309 – 340. Magró, Montevideo.

[12] L. Fernández, R. Peña, F. Nava, A. Velázquez, Análisis de las propuestas de la enseñanza de la programación orientada a objetos en los primeros cursos, (2002). Recuperado el 20 de junio de 2014 de: http://bioinfo.uib.es/~joemiro/aenui/procJenui/Jen2002/Cac457_464.pdf

[13] E. Dijkstra, Cooperating sequential processes, (1965). Recuperado el 5 de febrero de 2014 de http://www.dc.uba.ar/materias/so/2010/verano/descargas/articulos/dijkstra1965_semaforos.pdf/view

[14] C. Hoare, Communicating Sequential Processes, (2004). Recuperado el 12 de diciembre de 2013 de: http://www.usingcsp.com/cspbook.pdf

Dirección de Contacto del Autor/es:

Alejandro Miños
INET- Guatemala 1172
Montevideo
Uruguay
e-mail: alejandromifa@gmail.com

Juan Sales
Dirección Línea 1
INET- Guatemala 1172
Montevideo
Uruguay
e-mail: juanp.sales.gonzalez@gmail.com

Alejandro Miños Fayad: Profesor de Informática (I.N.E.T. – C.F.E.), Analista en Computación (Facultad de Ingeniería - UDELAR). Profesor de Didáctia en I.N.E.T. y de Progrmación y Sistemas Operativos en C.E.T.P.

Juan Pablo Sales González: Analista de Sistemas especializado en Sistemas Operativos y Redes. Profesor de Sistemas Operativos y Redes en I.N.E.T. y C.E.T.P. (Tecnólogo en Informática)