# Experiencias universitarias de aula en la introducción a la programación

## *University Classroom Experiences in Introduction to Programming*

*Recibido: 26 de octubre de 2020 / Aprobado: 30 de noviembre de 2020*

**Alexander Castro-Romero**[a]
UNIVERSIDAD PEDAGÓGICA Y TECNOLÓGICA DE COLOMBIA
alexander.castro01@uptc.edu.co

**Juan Sebastián González-Sanabria**[b]
UNIVERSIDAD PEDAGÓGICA Y TECNOLÓGICA DE COLOMBIA
juansebastian.gonzalez@uptc.edu.co

## Resumen

La programación y la algoritmia es transversal a una gran cantidad de asignaturas de las carreras de ingeniería, en particular, en aquellas carreras de computación, informática y sistemas. Son diversas las razones por las cuales sus técnicas y su aprendizaje no son adquiridos satisfactoriamente por los estudiantes. Algunas dependen de los conocimientos previos, motivación o creatividad del estudiante, y otras son atribuidas a la práctica docente o al nivel de abstracción que obliga la disciplina. La combinación de estas causas potencia las dificultades y producen un gran desgranamiento de los estudiantes en los cursos de los primeros años. La complejidad manifiesta de enseñar programación es constantemente abordada a través de diferentes estrategias. En este trabajo se analiza esa práctica y se observan las técnicas utilizadas. Metodológicamente, se optó por la observación participante con pruebas de control. La población estaba constituida por diversos grupos que fueron sometidos a distintos contextos controlados con el fin de determinar su comportamiento y nivel de evolución. Quedaron en evidencia buenas prácticas y algunas muy débiles, tanto a nivel del docente como de los estudiantes. La conclusión permitió corroborar algunos supuestos previos al estudio, poniendo en evidencia que el mayor logro se obtuvo cuando el docente ocupó el rol de mentor por encima de quien solo tiene el propósito de transmitir un saber descontextualizado y despersonalizado. El mayor beneficio se observó cuando el docente se constituyó en asesor del proceso individual de cada alumno. Esto generó retroalimentación continua de los participantes. Además, se pudo determinar la importancia de la motivación, en particular, en el proceso de aprendizaje, por lo que se compartieron mecanismos complementarios al aula para desarrollar esa motivación como «Hackathons» o maratones de programación.

**Palabras clave:** programación; algoritmos; actividades curriculares; educación informática.

## Abstract

Programming and algorithms are transversal to many subjects in engineering careers, particularly in those in computing, informatics and systems. There are various reasons why its techniques are not satisfactorily acquired by students. Some depend on the prior knowledge, motivation or creativity of the student, and others are attributed to teaching practice or the level of abstraction required by the discipline. The combination of these causes increases the difficulties and produces a great loss of students in the courses of the first years. The manifest complexity of teaching programming is constantly addressed through different techniques. In this work this practice is analyzed, and these techniques are observed. Methodologically, participant observation with control tests was chosen.

a. M. Sc. Universidad Pedagógica y Tecnológica de Colombia. Profesor a tiempo completo en la Universidad Pedagógica y Tecnológica de Colombia (Tunja-Boyacá, Colombia). alexander.castro01@uptc.edu.co. ORCID: https://orcid.org/0000-0001-9469-5445

b. M. Sc. Universidad Pedagógica y Tecnológica de Colombia. Profesor a tiempo completo en la Universidad Pedagógica y Tecnológica de Colombia (Tunja-Boyacá, Colombia).juansebastian.gonzalez@uptc.edu.co. ORCID: https://orcid.org/0000-0002-1024-6077

ISSN (en línea): 1814-4152 / Sitio web: http://cuaderno.pucmm.edu.do

CÓMO CITAR: Castro-Romero, A. y González-Sanabria, J. (2021). Experiencias universitarias de aula en la introducción a la programación. Cuaderno de Pedagogía Universitaria, Vol. 18, n.º 35, enero-junio, pp. 85-94

The population consisted of various groups that were subjected to different controlled contexts to determine their behavior and level of evolution. Good and bad practices were evident both at the level of the teacher and the students. The conclusion allowed to corroborate some assumptions prior to the study, showing that the greatest achievement was obtained when the teacher occupied the role of mentor over the one who only had the purpose of transmitting decontextualized and depersonalized knowledge. The greatest benefit was observed when the teacher became an advisor to the individual process of each student. This generated continuous feedback from the participants each on their role. In addition, it was possible to determine the importance of motivation, particularly in the learning process, so complementary mechanisms were shared with the classroom to develop this motivation, such as «Hackathons» or programming marathons.

**Keywords:** programing; algorithms; curricular activities; computer education.

## 1. Introducción

Existen diversos modelos pedagógicos, con enfoques y herramientas distintas. No obstante, no hay un modelo que garantice la formación integral en todas las disciplinas. La construcción del conocimiento, en particular, en áreas de la ciencia o de disciplinas que requieran gran nivel de abstracción, no resuelven la formación aferrándose a un único modelo, especialmente en el área de la programación donde, incluso, investigadores han detectado mayor dificultad para adaptar prácticas de enseñanza.

La existencia de estrategias tradicionales para la enseñanza de la programación solo es usada por el 17% de los desarrolladores, pues, por ejemplo, según StackOverflow (2018), el 90% prefiere el autoaprendizaje formal y el 48% lo complementa con cursos en línea. Es por ello por lo que la complejidad manifiesta de enseñar programación es constantemente abordada a través de diferentes técnicas como: clases teórico-prácticas, clases magistrales usando herramientas de programación por bloques, plataformas interactivas como Code Academy, cursos virtuales basados en videos y lecturas como Udacity, servicios de mentores, participación en proyectos de software libre, clases interactivas usando hardware o, incluso, campamentos intensivos como IronHack, clases invertidas, entre otras.

También es de resaltar que existen las instituciones universitarias que en su oferta académica tienen programas que incluyen dentro de sus contenidos asignaturas de programación (IEEE-ACM, 2016), sin embargo, estas tienen la particularidad de que, por su pluralidad en la enseñanza, no pueden enfocarse en el desarrollo de un modelo o una metodología puntual para la enseñanza de la programación. Por lo anterior, y en miras de mejorar la educación ofrecida a los estudiantes, se hace necesario contar con prácticas alternativas que permitan enseñar programación, para coadyuvar a que las instituciones sean eficaces en un ambiente cada vez más complejo y demandante.

Analizando el panorama académico, las estrategias para enseñar programación tienen diferentes enfoques. Algunas se concentran el proceso de enseñanza. Por ejemplo, se plantea el uso de una herramienta para modelar y validar los algoritmos antes del iniciar a programar (Moroni y Señas, 2005), lo que “disminuye la ansiedad por el uso de la computadora, facilita la nivelación entre alumnos y fomenta la realización del chequeo del algoritmo, como etapa previa a la codificación del mismo”. El estudio destaca que esta estrategia es buena, pero toma bastante tiempo y el grupo de muestra es limitado. Por su parte, Liu, Tong, y Yang (2018) plantean una alternativa similar, pero usando mapas conceptuales.

Otros estudios se centran en los temas que se deberían estudiar en las asignaturas de programación y además mencionan algunas ideas interesantes que apoyan lo anteriormente dicho: “algunas de las razones son que no existe un único método para la resolución de algoritmos, así como tampoco un enfoque didáctico para materias introductorias que se haya impuesto por sobre otros o demostrado una indiscutible efectividad” (Ferreira Szpiniak y Rojo, 2016). Los autores presentan diez unidades que aplican en un curso de introducción a la algoritmia y programación dejando claro que antes de este primer curso, los estudiantes se nivelan en un curso de ingreso donde se trabaja en resolución de problemas, lógica y matemática. Por último, exponen una metodología donde año tras año se revisa la estrategia planteada, se analizan aciertos y errores, se utilizan estadísticas de deserción, calificaciones y encuestas con los estudiantes y se plantean mejoras. (Ferreira Szpiniak y Rojo, 2016),

En cambio, Vera et al. (2016) se centran en el proceso de revisión y en cómo motivar a los estudiantes. Los autores destacan dos

ideas importantes: “la capacidad de elaborar estrategias para la resolución de los problemas (algoritmos) es una capacidad que se va forjando lentamente a medida que se incorporan nuevas herramientas que brindan los lenguajes de programación” y “se pone en evidencia la falta de dedicación de tiempo por parte de los alumnos al no realizar ejercitación por su cuenta”, y se propone una estrategia que emplea la técnica de gamificación de la cual ya son conocidas sus ventajas. Como punto débil se menciona que el docente requiere de bastante tiempo para plantear los ejercicios.

En otro tipo de estudios se focaliza la atención en identificar los problemas que tienen los estudiantes a la hora de aprender a programar. Para ello, se encuesta a los estudiantes acerca de los procesos y temas más complejos. Como resultado, se encontró que las actividades que implican mayor dificultad corresponden al desarrollo de la prueba de escritorio, la identificación de errores y la corrección de estos (Casas y Vanoli, 2007). Este estudio mostró también que los temas que presentan mayor dificultad son la recursividad y el manejo de memoria dinámica.

Por último, se destaca la tendencia de algunos estudios al mostrar la experiencia de usar programas que combinan el desarrollo del pensamiento computacional con la programación visual (Lye y Koh, 2014), con herramientas como, por ejemplo, Logo, Scratch, Alice, entre otros, e inclusive crean sus propios softwares (Pérez-Tavera, 2015; Zuleta Medina y Chaves Torres, 2011). Sin embargo, esta tendencia plantea muchos problemas al aplicarse en un entorno universitario, entre ellos: muchos programas están diseñados para niños de edades tempranas y los estudiantes universitarios pierden el interés rápidamente; demandan mucho tiempo; sirven para explicar solo conceptos básicos; se hace difícil la transición a modelar en una notación formal, escribir código y probarlo; el estudiante se distrae fácilmente y, aunque desarrollan la lógica, limitan la creación de soluciones reales, son de un nivel muy alto y crean un modelo mental inexacto en el estudiante acerca de cómo funcionan los programas y este modelo se vuelve difícil de cambiar (Gallego-Durán et al., 2018).

En este último punto, aunque se han encontrado muchas experiencias de docentes que aplican metodologías y herramientas obteniendo buenos resultados en términos de disminución de la deserción, nivel de motivación, aumento del aprendizaje (Erol y Kurt, 2017), surgen varias preguntas: ¿por qué estas experiencias no han tenido continuidad?, ¿por qué los resultados siempre son positivos si hay otros factores de deserción adicionales a la metodología usada por el docente? ¿el interés de los docentes por demostrar que su metodología sirve será un problema a la hora de evaluarla objetivamente?

Teniendo en cuenta los tipos de estudios tratados y sus enfoques, este trabajo integra recomendaciones generales y no busca centrarse en un aspecto en particular, esto para que cada docente pueda adaptar las prácticas que considere oportunas. Por ello, sintetiza las experiencias de un grupo de docentes universitarios recopiladas durante cinco años de enseñanza del nivel introductorio de programación en una universidad pública en el que participaron más de 450 estudiantes. Este trabajo plantea unas recomendaciones para enseñar el curso introductorio a la programación en un ambiente universitario, tomando como punto de partida las clases presenciales tradicionales e integra otros enfoques y recomendaciones basados en la experiencia de los autores.

En el trabajo se expone el resultado de un aprendizaje mecánico que se llevaba a cabo inicialmente, antes de incluir innovaciones en el proceso de enseñanza-aprendizaje. Seguidamente se explican las estrategias que los docentes consideraron exitosas para los problemas más comunes. Finalmente, se presentan las conclusiones a las que hemos abordado con el estudio llevado a cabo.

## 2. Metodología

Para este estudio se toma como referencia el programa de Ingeniería de Sistemas y Computación de la Universidad Pedagógica y Tecnológica de Colombia, una universidad pública ubicada en la cuidad de Tunja (a dos horas de la capital del país) que cuenta con aproximadamente 25.000 estudiantes. El programa de Ingeniería de Sistemas y Computación incluye dentro del plan de estudios cuatro niveles de programación, este trabajo se centrará en las experiencias con el primer nivel “Algoritmos y programación”. Entonces, con el ánimo de mejorar el proceso de enseñanza-aprendizaje en este programa, desde el año 2015 hasta el año 2019, se han realizado ajustes graduales en el proceso que han llevado

a la construcción de un conjunto de principios, herramientas y recomendaciones para la enseñanza de la programación.

Este proceso se realizó cada semestre con una población promedio de 45 estudiantes para un total aproximado de 450 estudiantes divididos en cada semestre en 3 grupos de 15 estudiantes. Estos estudiantes provienen de diferentes regiones, contextos socioeconómicos y llegan a la universidad con diferentes niveles de conocimientos. La asignatura de Programación es el primer acercamiento al programa y cuenta con seis horas semanales que se imparten durante dieciséis semanas con un enfoque teórico practico. Cada grupo de estudiantes es dirigido por un docente diferente.

El estudio contó con dos etapas principales, en la primera, a través de una metodología descriptiva, se buscó identificar las dificultades más significativas del proceso de enseñanza-aprendizaje de la asignatura de Algoritmos y Programación, tanto a nivel de los docentes como a nivel de los estudiantes. Esto se logró a través entrevistas y un grupo de discusión conformado por los docentes del área. De cada etapa se buscaba obtener un conjunto de incidencias u oportunidades de mejora en la enseñanza del curso introductorio de Programación.

Entonces, se caracterizó el proceso seguido tradicionalmente por el programa para la enseñanza de las asignaturas de programación, el cual consistía, *groso modo*, en que el docente explica un tema, crea el código de lo explicado y los estudiantes lo transcriben. Al final, el docente evalúa con ejercicios similares a los propuestos en clase, convirtiéndose así en un aprendizaje mecánico. Como resultado de esta etapa, se hallaron algunos incidentes expuestos en el tercer acápite. Se destaca que algunos de los problemas encontrados son de índole organizacional, pero que vale la pena destacarlos, pues afectaban directamente el proceso educativo de los estudiantes. También se resalta que uno de los problemas más graves detectado fue que no había consenso en los temas básicos de la asignatura, por ello se construyeron unos contenidos mínimos que se pueden verificar en la tabla 1.

En una segunda etapa los docentes del área de programación siguieron un modelo iterativo, donde cada semestre se realizaban las siguientes actividades: se introducían actualizaciones y cambios graduales, acompañados de diversas estrategias que buscaban mejorar el aprendizaje de los estudiantes (evidenciado en aspectos cualitativos o difíciles de medir como la calidad de los proyectos realizados, los comentarios finales del curso, el progreso general del grupo durante el curso, la respuesta a las evaluaciones, entre otros) y al finalizar cada semestre, se evaluaba la efectividad de cada actualización o estrategia aplicada (a través de una retrospectiva de las notas de campo recolectadas, una autoevaluación critica de los docentes y un análisis de los resultados alcanzados por los estudiantes) y de ser positivo el resultado obtenido, se consolidaba su inclusión formal para el siguiente semestre, además de que se discutían nuevas estrategias. Este proceso estuvo acompañado de reuniones semanales para hacer seguimiento a los cambios planteados. Para ello, se tomó como referencia el marco de trabajo Scrum que permite gestionar el trabajo colaborativo entre equipos, por medio de mecánicas como lista de tareas y reuniones de retrospectiva para analizar el trabajo realizado.

La adaptación de Scrum consistió en que al inicio del semestre se establecía una lista de tareas que consistían en estrategias o actualizaciones a las clases, en cada inicio de Sprint (periodo de trabajo de dos semanas) un profesor proponía qué estrategias iba a aplicar durante ese tiempo y los compañeros le daban sugerencias de mejora, al finalizar las dos semanas, se discutía el resultado de la aplicación de la estrategia, por ejemplo con la estrategia: “Estudiar herencia con un ejercicio donde el estudiante deba construir botones personalizados con diferentes formas y comportamiento”, el docente manifestó que el hecho de crear familias de componentes gráficos brindó mejores resultados para el tema de herencia que los ejemplos típicos de animales (algo similar a lo que se hace en una revisión del Sprint), entonces todos los docentes compartían sus experiencias, aciertos y problemas encontrados y se llegaba a la conclusión de dejar algunas estrategias exitosas para repetirlas el próximo semestre y este proceso se repetía durante todo el semestre (como en la reunión de retrospectiva de Scrum).

Finalmente, se hace necesario aclarar que este estudio no mostrará en sus resultados estadísticas de deserción, calificaciones o encuestas a estudiantes, pues la enseñanza es un proceso que depende de variables que no siempre son cuantificables. Por lo anterior, los resultados obtenidos están basados en la experiencia de los docentes en los cursos y la sensación de mejora percibida en los estudiantes. En el apartado cuatro se presentan las estrategias que los docentes consideraron exitosas para los problemas más comunes.

## 3. Resultados

Por medio de observación directa, encuestas y foros de discusión, entre otros, se evidencia la necesidad de mejorar el nivel académico del área de programación y se destacan los siguientes problemas:

- Bajos conocimientos en ciencias de la computación y desconocimiento del perfil profesional: muchos de los nuevos estudiantes poseen falencias en habilidades como lógica, uso de herramientas computacionales (hardware y software) y, en general, en un proceso de aprendizaje ordenado y exigente. Además, algunos estudiantes no tienen claro de qué se trata el programa, ya que ingresaron influenciados por factores externos o con una idea equivocada del perfil profesional.

- Segmentación de los grupos: se identificó que los estudiantes se dividían en tres grupos: los que presentaban problemas con algún tema, los que conocen previamente el tema o que lo asimilan fácilmente y los estudiantes que iban al ritmo de los contenidos. Los dos primeros grupos perdieron interés rápidamente y/o disminuyeron su rendimiento en las asignaturas, llegando a que algunos, incluso, cancelaran su inscripción en las mismas.

- Heterogeneidad de contenidos: debido a la contratación por horas-clase de algunos docentes, se fomentaban situaciones como cambios en los contenidos de las asignaturas y en el lenguaje de programación, al intentar adaptar los contenidos a sus conocimientos y no ellos adaptarse a los contenidos programáticos mínimos definidos para el programa (algunos inclusive no conocían el lenguaje que iban a enseñar). Lo anterior, promovió en los estudiantes las malas prácticas de programación, el uso de herramientas inadecuadas y el aprendizaje de temas que son avanzados para el nivel de los estudiantes y luego creaban confusión (por ejemplo, bases de datos).

- Impactos negativos a futuro: siendo la programación el núcleo del programa, docentes de asignaturas de semestres posteriores manifestaron que los estudiantes llegaban sin algunas de las habilidades básicas de programación, no solo a nivel de conocimientos, sino también en habilidades "blandas" (por ejemplo, trabajo en equipo y bajo presión).

- Dificultades para la adaptación: a pesar de que los estudiantes tenían conocimientos básicos de programación y habían trabajado con un lenguaje durante dos años (Java), presentaron dificultades para programar en otros lenguajes.

- Poca visibilidad de resultados: al finalizar los cursos, los estudiantes no evidenciaron un portafolio de proyectos propios. En su lugar disponían de una colección de códigos creados por el docente, copiado de internet o fragmentos de código por temáticas, pero no tenían la oportunidad de hacer proyectos a mayor escala y que fueran incrementales en la integración de los conceptos vistos.

- Miedo al cambio por parte de los docentes: algunos docentes llevaban dictando materias del área por varios años, usando los mismos recursos y estrategias y se mostraban renuentes a cambiarlos. Incluso algunos llevaban cometiendo errores mucho tiempo que eran repetidos por sus estudiantes y ellos de buena fe creían estar en lo correcto.

- Otros aspectos no directamente académicos: por ejemplo, los horarios de clase (es difícil para ellos concentrarse un viernes en la tarde, ya que la mayoría viven fuera de la cuidad y deben viajar a sus hogares); la asignación académica, pues en los semestres que se ve las materias de programación, los estudiantes no ven ninguna materia de la carrera sino materias del núcleo común de ingeniería, como física y cálculo que demandan mucho tiempo; problemas en el aspecto económico, social y hasta psicológico. Por último, se resalta que algunos estudiantes simplemente no tienen interés en estudiar, no solo programación, sino en términos generales y debido a las políticas laxas a nivel escolar e incluso universitario, siguen asistiendo a la universidad, pero sin un interés real en aprender.

Es pertinente aclarar que, pese a las debilidades identificadas, los estudiantes del programa se han caracterizado y son reconocidos a nivel empresarial por tener un destacado nivel de programación, sin embargo, parte de este fenómeno se daba más por la capacidad de auto capacitación que por el aprendizaje adquirido en la Universidad.

## 4. Debilidades encontradas y recomendaciones

En este nivel se asume que el estudiante no tiene ningún conocimiento de programación,

por lo tanto, los contenidos mínimos para esta asignatura de “Algoritmos y programación” son los registrados en la Tabla 1, resaltando que algunos de los temas son transversales, es decir, se trabajan durante todo el semestre, como es el caso de algoritmos, abstracción mediante objetos y codificación.

*Tabla No. 1*: Contenidos mínimos para la asignatura de algoritmos y programación

| TEMA | ENFOQUE |
|---|---|
| Algoritmos | Lenguaje natural, diagramas de flujo (se recomienda usar BPMN – Modelado y notación de procesos de negocio - como notación) y pseudocódigo |
| Configuración de entorno y proceso de compilación y ejecución | Procesos y herramientas, lenguaje de programación, editores y consola de comandos |
| Codificación | Sintaxis del lenguaje, buenas prácticas y consejos generales |
| Tipos de datos | Relacionándolo con sistemas numéricos (por ejemplo, binario) |
| Operadores | Lógicos, aritméticos y unitarios |
| Abstracción mediante objetos | Clases, atributos (variables y constantes) y métodos (entradas y salidas), nociones de diagrama de clases UML (Lenguaje Unificado de Modelado) usando solo la relación de asociación |
| Condicionales | If y switch |
| Ciclos | For, while y recursividad básica |
| Introducción a vectores | Operaciones básicas (agregar, borrar, leer, buscar, reportes simples) |
| Creación de aplicaciones con interacción de usuario | Usando línea de comando y aplicando el patrón de MVP (Modelo Vista Presentador) |

Teniendo en cuenta las debilidades generales encontradas, se indagó a mayor profundidad con los estudiantes y docentes para detectar problemas específicos del aprendizaje de programación en este primer nivel, algunas de las más relevantes fueron:

- Al ser la primera y única asignatura del programa que los estudiantes cursan, en los contenidos se incluían temas de introducción a ciencias de la computación e institucionales que disminuyen el tiempo de clase, necesario para el desarrollo de los temas propios de la asignatura.

  **Estrategias implementadas:** se removieron los temas que podrían considerarse complementarios y se motivó a tratar otros temas de forma superficial y con estrategias más lúdicas. Por ejemplo, para el tema de Introducción a las ciencias de la computación, se planteó que cada área del programa fuera asignada a un grupo de estudiantes y estos hicieran una entrevista a un experto; para el tema de conversión de números decimales a binarios, se hizo uso un juego en línea; para el tema de los sistemas numéricos, se plantearon ejercicios como investigar los colores de la imagen de la universidad en formato hexadecimal. En este punto fue importante la inclusión de la metodología de clase invertida, donde el estudiante consultaba aspectos teóricos antes de la clase y se aprovechaba el tiempo en el salón para hacer preguntas y realizar ejercicios prácticos.

- El estudiante realizaba una búsqueda en Internet o libros de los ejercicios planteados y se limitaban a copiar el código encontrado, que en la mayoría de las ocasiones es de baja calidad.

  **Estrategias implementadas:** se dejaron de utilizar los ejercicios comunes para la introducción a la programación (por ejemplo, factorial de un número, serie Fibonacci, etc.), y se propusieron ejercicios reales y con componentes aleatorios, esto usando estrategias por parte del docente. En una de ellas, este elegía una noticia del día al azar, planteando un ejercicio del tema de la noticia, por ejemplo, en una noticia de la restricción de vehículos usando el último dígito de la placa, se explicó el tema de condicionales, proponiendo un ejercicio donde se debía construir un método que recibía como parámetro la placa del vehículo y el día de la semana y se debía validar si el vehículo podía transitar o no ese día. Este enfoque basado en mini proyectos dio mejor resultado el de lista de ejercicios clásicos pues el estudiante estaba motivado por el problema y se identificaba con el contexto lo cual hacía que se comprometiera con la solución, también disminuyo los niveles de plagio pues estos proyectos no están resueltos en otra parte.

- Los docentes usaban diferentes mecanismos para gestionar la interacción del usuario con los programas, lo que llevaba a la producción de código con malas prácticas de programación, en especial, violación de principios básicos de la programación

orientada a objetos. Esto hacía que el estudiante no se concentrara en la lógica de programación, sino en las particularidades del lenguaje, como son la construcción de interfaces de usuario y realizar la unión de estas dos capas (lógica y vista). Muchos lo lograban, pero adquiriendo malas prácticas.

**Estrategias implementadas:** se definió que, para este nivel, la interacción con el usuario estaba limitada el uso de entrada y salida de datos por la consola del sistema, esto permite que el estudiante se concentre en la lógica del problema y se le inculque los principios de única responsabilidad y separación de asuntos, a través de una implementación básica del patrón MVP. Este tema se trata en la segunda mitad del curso. Al inicio, los datos se pasan como parámetro a los métodos directamente desde el método principal.

- Código "espagueti". Al estudiar el código de los estudiantes, se encontró que, a pesar de funcionar, este no cumplía con prácticas mínimas de calidad, esto nos llevó a buscar la raíz del problema. La conclusión fue que algunos docentes eran los que transmitían o dejaban pasar malas prácticas de programación pensando que estas después se corregirían. Algunas de estas prácticas era que tenían problemas con la asignación de nombres, poco respeto por las guías de estilo del lenguaje, uso de variables y métodos estáticos donde no se deberían aplicar (solo para evitar explicar cómo se crea la clase), uso de números mágicos, métodos muy largos, uso del método *main* para colocar la lógica de una función en vez de encapsularla en un método, entre otros denominados "olores de código" (Fowler, 2012).

  **Estrategias implementadas:** la solución vino de la mano de enseñar buenas y mejores prácticas desde el inicio y codificar en inglés. Este proceso se torna complejo, pero vale la pena ya que el estudiante se acostumbra y el código que realiza tiene calidad.

- Dependencia de las herramientas. Se identifica que los estudiantes adquieren malas prácticas al usar IDE (Entornos integrados de desarrollo), debido a que estas herramientas generan mucho código automáticamente que es usado sin comprenderse debidamente. Adicionalmente, muchas de las funciones evitan que el estudiante entienda procesos básicos que pueden ser realizados manualmente (por ejemplo, el proceso de compilación y ejecución), además, evitan el desarrollo de habilidades como el uso de la consola de comandos que puede llegar a ser de utilidad más adelante.

  **Estrategias implementadas:** teniendo en cuenta que el estudiante debe aprender a programar y no a usar una herramienta en específico, se propone usar durante la primera mitad del curso solo un editor de texto básico y realizar el proceso de compilación por consola, durante la segunda parte del curso, se sugiere usar un editor de texto avanzado (con funciones básicas como resaltado de sintaxis y autocompletado simple).

- Errores comunes, repetidos continuamente.

  **Estrategias implementadas:** es deber del docente anticiparse a los errores comunes y hacer entender que no está mal que el estudiante tenga un error en su código, lo que sí está mal es tener el mismo error varias veces. En este punto, se recomienda hacer énfasis en la división de tareas que el estudiante vaya trabajando en un método a la vez, compile y vea el resultado, para así volver la programación un proceso iterativo e incremental. Para ello es bueno hacer una lista de los errores comunes (Altadmri y Brown, 2015) y publicarla junto con su solución o tener una lista de chequeo con recomendaciones antes de ejecutar el código, por ejemplo, los objetos se comparan usando *equals* y los datos primitivos con ==, el operador = solo se usa para asignación.

- Ansiedad por usar el computador para codificar. El estudiante prefiere codificar a modelar y por su afán, comete errores lógicos y gasta mucho tiempo.

  **Estrategias implementadas:** este es un punto delicado pues no hay que llegar a prohibir al estudiante codificar, en cambio es necesario guiarlo a través del proceso de desarrollo de la solución. Para ello, se pueden plantear ejercicios de solo modelado que sean un reto para él, por ejemplo, intentar modelar algún software real. Por otra parte, mostrar plantear un ejercicio con cierta complejidad y animar al estudiante a llegar a la solución solo codificando, luego otro ejercicio donde se deba realizar el diseño de la solución y que el mismo concluya las ventajas del segundo enfoque.

- Dificultad en abstraer problemas y proponer soluciones debido a los pocos conocimientos.

  **Estrategias implementadas:** el docente generalmente quiere ampliar un ejercicio, pero se ve en la necesidad de explicar nuevos temas ante las dudas de los alumnos. Por ejemplo, se está haciendo un software para gestionar las calificaciones de un estudiante, pero aún no se ha visto el tema de vectores y el estudiante quiere hacer que el programa funcione con varias notas. Para ello se puede proponer guardar las notas en una variable de tipo String separadas por comas (aclarando que más adelante se estudiará la manera correcta de hacerlo), de esta manera, los ejercicios se vuelven más dinámicos y el estudiante va repasando sin necesidad de confundirlo con temas nuevos. Otro ejemplo de esto es la generación de reportes. El docente debe ser creativo para aprovechar los conocimientos básicos de los estudiantes y obtener mejores resultados. En vez de hacer un método que muestre el resultado de una encuesta en números (10%), usar los caracteres ASCII (American Standard Code for Information Interchange) para visualizar el mismo resultado de manera diferente (<|_____> %). También se debe motivar al estudiante a inventar sus propios reportes usando únicamente los temas vistos, es decir, sin recurrir a librerías externas, esto con el fin de desarrollar la lógica.

## 5. Referentes externos

Adicional a las incidencias encontradas, el grupo de docentes investigó tendencias en la enseñanza que fueron aplicadas exitosamente. En primera medida se destaca, la aplicación de la metodología de clase invertida fue muy productiva (Arellano et al., 2015). En ella el estudiante consulta la teoría en horario extra clase e inicia cada sesión con un conocimiento básico del tema a tratar (se resalta que es mejor solicitar a los estudiantes las consultas escritas a mano, esto para mejorar los niveles de aprendizaje), luego el docente refuerza estos conceptos y el estudiante los aplica en ejercicios prácticos con acompañamiento del docente y, por último, se proponen retos para mejorar los productos creados, lo que motiva al estudiante a seguir trabajando.

Adicionalmente, la aplicación de aprendizaje basado en proyectos permitió a los estudiantes mejorar sus habilidades blandas y estar motivados al finalizar cada periodo académico. Esto lo evidencian Halverson y Sheridan (2014) quienes proponen un movimiento de “creadores”, donde se haga, se comparta, se aprenda, se usen herramientas, se juegue, se participe y, por último, que el estudiante cambie su manera de ver el mundo. En este movimiento el estudiante crea proyectos funcionales, los comparte y trabaja de manera colaborativa. Para ello, también ha dado buenos resultados motivar al estudiante a que aprenda por sí mismo un nuevo lenguaje o tecnología, por esto se proponen actividades que otorgan bonificaciones en calificación final como: leer un libro de tecnología y llevar un diario; realizar un curso en línea de un lenguaje nuevo; contestar preguntas en foros de programación y hacer una contribución de código a un proyecto de software abierto.

También es interesante ver cómo algunas de las recomendaciones dadas ya han sido recopiladas en listas como las propuestas por Brown y Wilson (2018), en las que se dan los siguientes consejos para enseñar a programar. Entre los que se repiten están: programa en vivo, enseña solo un lenguaje, asigna tareas reales. Sin embargo, otros como dejar que los estudiantes hagan predicciones y el de usar ejercicios estructurados también se deben tener en cuenta. Por otra parte, algo que ha dado buenos resultados es crear eventos que motiven a los estudiantes a programar como “Hackatones”, exposición de trabajos o maratones de programación (Fracchia et al., 2014).

## 6. Conclusiones

Se puede concluir que el trabajo en equipo entre los docentes del área es fundamental para estar en constante mejoramiento. Las revisiones de contenidos y conceptos deben ser permanentes. Se recomienda conformar un equipo de trabajo donde se expongan temas, dudas y las experiencias de clase. Algo que permitió organizar mejor el trabajo de los docentes fue utilizar el marco de trabajo Scrum para dar seguimiento a las estrategias, ya que este permite ordenar el trabajo en equipo y ofrece un seguimiento de los avances logrados.

Una recomendación que podemos dar a los docentes de introducción a la programación es que no intenten abarcar muchos temas para que no caigan en un aprendizaje superficial saturado de contenido, sino que reten a los estudiantes con proyectos que los hagan integrar sus saberes adquiridos, y que estos proyectos sean contextualizados a partir de las situaciones cotidianas o de la región donde viven. En definitiva, se trata de no complejizar la clase

con muchas herramientas, sino centrarse en enseñar desde un comienzo buenas prácticas de programación.

Es necesario, también, reforzar el orden de enseñanza priorizando el modelado y diseño de los algoritmos antes del iniciar a programar (Moroni y Señas, 2005) y cuando los tiempos de clase sean limitados, reforzar con mapas conceptuales (Liu et al., 2018). Sin embargo, se hace necesario precisar que no se debe abusar del uso de herramienta computacionales para explicar estos primeros temas, pues pueden limitar la creación de soluciones reales (Gallego-Durán et al., 2018).

Adicionalmente, es importante destacar los buenos resultados que se obtuvieron de aplicar técnicas como las propuestas por Arellano et al. (2015) de clase invertida, propiciando una participación más activa del estudiante. Así como el uso de herramientas complementarias que coadyuvan al refuerzo del aprendizaje en el aula y evitan la monotonía (Erol y Kurt ,2017).

Finalmente, se destaca que el rol de profesor debe orientarse al de mentor. En vez de tratar de impartir un saber a un grupo, el docente debe asesorar el proceso individual de cada estudiante. Para ello, es necesario estar en constante actualización y en interacción permanente con los estudiantes para ofrecer realimentación de los trabajos o pruebas realizadas y guiándolos a través de estrategias colaborativas para nivelar al grupo, de modo tal que se logre motivar a los que vayan quedándose atrasados con respecto al grupo.

## Referencias bibliográficas


Altadmri, A. y Brown, N. C. (2015). 37 million compilations: Investigating novice programming mistakes in large-scale student data. En *46th ACM Technical Symposium on Computer Science Education de ACM*, Kansas City, USA. https://doi.org/10.1145/2676723.2677258

Arellano, N. M., Aguirre, J. F. y Rosas, M. V. (2015). Clase invertida: una experiencia en la enseñanza de la programación. En *X Congreso sobre Tecnología en Educación & Educación en Tecnología*, Universidad Nacional de San Luis, Argentina. http://sedici.unlp.edu.ar/bitstream/handle/10915/49121/Documento_completo.pdf?sequence=1&isAllowed=y

Brown, N. y Wilson, G. (2018). Ten quick tips for teaching programming. *PLoS computational biology*, 14 (4), e1006023. https://doi.org/10.1371/journal.pcbi.1006023

Casas, S. y Vanoli, V. (2007). Programación y Algoritmos: Análisis y Evaluación de Cursos Introductorios. En *IX Workshop de Investigadores en Ciencias de la Computación de la Red de Universidades con Carreras de Informática*, Argentina.

Erol, O. y Kurt, A. A. (2017). The effects of teaching programming with scratch on pre-service information technology teachers' motivation and achievement. *Computers in Human Behavior*, 77, 11-18. https://doi.org/10.1016/j.chb.2017.08.017

Ferreira Szpiniak, A. y Rojo, G. A. (2016). Enseñanza de la programación. *Revista Iberoamericana de Tecnología en Educación y Educación en Tecnología*, 1(1), 8.

Fowler, M., Beck, K., Roberts, D. y Gamma, E. (2012). *Refactoring: Improving the Design of Existing Code*. Reino Unido: Addison-Wesley.

Fracchia, C. C., Kogan, P., Alonso, A. C., Godoy, I., y López, L. M. (2014). Realización de torneos de programación como estrategia para la enseñanza y el aprendizaje de programación. En *XX Congreso Argentino de Ciencias de la Computación*, Argentina.

Gallego-Durán, F. J., Satorre-Cuerda, R., Compañ-Rosique, P. y Villagrá-Arnedo, C. (2018). Explicando el bajo nivel de programación de los estudiantes. *ReVisión*, 11 (1), 33-42.

Halverson, E. R. y Sheridan, K. M. (2014). The Maker Movement in Education. *Harvard educational review*, 84 (4), 495-504.

IEEE - ACM (2016). *Computer Engineering Curricula*, 2016. https://www.acm.org/binaries/content/assets/education/ce2016-finalreport.pdf

Liu, Y., Tong, Y., y Yang, Y. (2018). The Application of Mind Mapping into College Computer Programming Teaching. *Procedia Computer Science*, 129, 66-70. https://doi.org/10.1016/j.procs.2018.03.047

Lye, S. y Koh, J. H. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12?. *Computers in Human Behavior*, 41, 51-61.

Moroni, N. y Señas, P. (2005). Estrategias para la enseñanza de la programación. En *Primeras Jornadas de Educación en Informática y TICS de la Red de Universidades con Carreras en Informática*, Argentina. http://sedici.unlp.edu.ar/bitstream/handle/10915/18901/52.pdf?sequence=1

Pérez-Tavera, I. H. (2015). Scratch en la educación. *Vida Científica Boletín Científico de la Escuela Preparatoria*, 13, 35-36.

StackOverflow (2018). *Developer Survey Results*. https://insights.stackoverflow.com/survey/2018/#education

Vera, P. M., Moreno, E. J., Rodríguez, R. A., Vázquez, M. C. y Valles, F. E. (2016). Aplicación de Técnicas de Gamificación para la Enseñanza de Programación a Alumnos de Primer Año de Ingeniería. En *XI Congreso de Educación en Tecnología y Tecnología en Educación de la Red de Universidades con Carreras en Informática*, Argentina. http://sedici.unlp.edu.ar/handle/10915/54653

Zuleta Medina, A. y Chaves Torres, A. (2011). Uso de herramientas informáticas como estrategia para la enseñanza de la programación de computadores. *Revista Unimar*, 57, 23-32.