Propuesta de Protocolo de Análisis de Dispositivos de Enseñanza de Programación

Mauricio Dávila[1], Marisa Panizzi[1], Darío Rodríguez[2], Ramón García-Martínez[2]
1. Programa de Maestría en Ingeniería de Sistemas de Información. UTN FRBA, Argentina
2. Grupo de Ingeniería de Espacios Virtuales de Trabajo y Grupo de Investigación en Sistemas de Información. Departamento de Desarrollo Productivo Tecnológico. Universidad Nacional de Lanús
davilamr.80@gmail.com, djhr_1977@yahoo.com.ar, rgm1960@yahoo.com

Resumen

En los cursos iniciales de programación en el nivel universitario, uno de los vectores de análisis de la deserción lo constituye las debilidades de los dispositivos de enseñanza utilizados. En una investigación exploratoria se detectaron diferencias de criterios sobre la estructuración de los contenidos de los cursos analizados. En este trabajo se presenta un protocolo estructurado en tres ejes: contenidos, didáctica, y herramientas, para el análisis de un curso introductorio del área de programación. Se presenta una prueba de concepto que permite validar la utilidad del protocolo propuesto.

Palabras claves: Algorítmia, Dispositivos de Enseñanza, Protocolo de Análisis.

1. Introducción

El pensamiento computacional se basa en resolver problemas haciendo uso de conceptos de informática, desarrollar esta habilidad es la tarea principal de las asignaturas de introducción a la programación en las carreras universitarias. En general, aunque se han creado lenguajes cada vez más cercanos al lenguaje humano, la programación no resulta ser una materia intuitiva ni de fácil comprensión y, por lo tanto, con frecuencia tienen altas tasas de deserción. Según [Byrne y Lyons, 2001] existe una gran cantidad de estudiantes que alcanzan competencias en otros temas y que no logran alcanzar el éxito en las materias relacionadas a la programación. La preocupación por los cursos iniciales de programación en el nivel universitario ha llevado a muchos investigadores a analizar las causas que subyacen en esta problemática desde distintos enfoques.

Algunos autores abordan el tema analizando los problemas de comprensión de conceptos por parte de los estudiantes, [Batman y Mayer, 1983] examinaron conceptos erróneos relacionados con las sentencias de programas escritos en el lenguaje BASIC, encontrando que muchos estudiantes tenían comprensiones erróneas o falsas ideas. [Spohrer y Soloway, 1986] examinaron la causa de los errores cometidos al momento de programar buscando determinar si éstos son producto de conceptos erróneos acerca de la semántica de los lenguajes de programación, llegaron a la conclusión que los errores son más propensos a surgir de falencias en la lectura y el análisis de las especificaciones. [Brito y Sá-Soares, 2014] sostienen que los alumnos tienden a sobreestimar su nivel de aprendizaje y esta percepción equivocada los lleva a fracasar.

En el trabajo de [Leone et al., 2014] se identifican múltiples causales que pueden llevar a la permanencia o abandono de un alumno y los agrupa según los siguientes factores:

i. Factores personales: características individuales como competencias desarrolladas, experiencias previas, vocación, limitaciones, dificultades.
ii. Factores estructurales: se consideran diversos elementos del ambiente universitario que pueden tener una importante influencia, como por ejemplo, medios utilizados; servicios brindados; infraestructura; sistemas informáticos.

iii. Factores académicos: se refieren a la propuesta formativa e incluye tanto las actividades curriculares como las prácticas docentes, reglamentos o actividades extra-curriculares.

iv. Factores sociales: hacen a la relación con los restantes actores, dado que a partir del ingreso el estudiante genera un nuevo mapa de vínculos y relaciones.

Muchas son las hipótesis que tratan de explicar el fenómeno de la deserción en los cursos iniciales de programación en el nivel universitario y por tratarse de una problemática compleja ninguna de estas interpretaciones agota el tema. Otros autores proponen soluciones de distinta índole para mejorar alguno de los causales de la deserción.
Una gran cantidad de estudios muestran que los entornos de aprendizaje basados en el contexto mejoran la participación de los estudiantes [Becker, 2001; Resnick et al., 2009; Sung et al., 2008], éstos se tratan de herramientas que le permiten al estudiante escribir código y observar de inmediato la ejecución. De esta manera, los estudiantes llegan a entender los conceptos abstractos de programación. Ejemplos de entornos de aprendizaje basado en el contexto son: Scratch [Resnick et al., 2009] y Karel [Becker, 2001].
Otra herramienta de aprendizaje basado en el contexto que ha ganado mucha atención últimamente es el uso de robots educativos; un ejemplo es el uso del robot Lego MindStorms [Klassner y Anderson, 2003].
Otra solución propuesta es la instrucción por pares, una técnica pedagógica para aumentar la participación en clase, en ésta los alumnos comienzan respondiendo a una pregunta de selección múltiple de manera individual y emitido su voto lo discuten en grupos para consensuar un voto grupal. [Porter, et al 2011; Simon et al, 2010]
Otra alternativa propuesta es la realización de prácticas de programación de manera colaborativa [Estácio, et al, 2015], éstas difieren principalmente en la forma y en el número de participantes que se asigna a la actividad. Dos ejemplos de estas prácticas son, la programación por parejas en la cual se busca promover la cooperación entre programadores [McDowell et al., 2006; Rimington, 2010] y la colaboración en equipos como puede ser Coding Dojo Randori [Rooksby et al., 2014] , existe un problema a resolver y un grupo de alumnos a los que se asigna la tarea, dos integrantes del grupo comienzan realizando programación de a pares y en intervalos regulares de tiempo uno de los dos integrantes es reemplazado, esta técnica provoca que todos los participantes deben prestar suma atención ya que desconocen en qué momento les tocará tomar el mando y continuar programando. Al igual que existen una gran número de hipótesis que intentan explicar el fenómeno de la deserción en los cursos iniciales de programación en el nivel universitario, existen un gran número de soluciones que se han propuesto hemos hecho referencias solo a algunas.
En este contexto, se delimita el problema planteado (sección 2), se propone un protocolo para dar solución al problema identificado (sección 3), se presenta una prueba de concepto del protocolo propuesto (sección 4), y se dan conclusiones preliminares (sección 5).

2. Delimitación del Problema

Uno de los vectores de análisis de los causales de deserción podría ser evaluar las fortalezas y debilidades de los dispositivos de enseñanza utilizados. En una revisión preliminar sobre los contenidos de los cursos iniciales de programación en el nivel universitario se detectaron diferencias de criterio sobre la estructuración de los mismos.
El problema que se aborda en esta comunicación es la definición de un protocolo que nos permita caracterizar aspectos de diseño de dispositivos de enseñanza utilizados en las materias introductorias de programación en el nivel universitario.

3.Propuesta de Protocolo

Para abordar el problema de análisis del dispositivo de enseñanza de Programación en un es-

tadio inicial hemos agrupado los aspectos en tres grupos: contenidos (sección 3.1), didáctica, contenidos (sección 3.2), y herramientas (sección 3.3). En la tabla 1 se sintetiza el protocolo propuesto.

Tabla 1. Protocolo de Análisis para Cursos Iniciales de Programación

Contenido	Algoritmos y Diseño	El concepto y propiedades de los algoritmos
		El papel de los algoritmos en el proceso de resolución de problemas
		Las estrategias de resolución de problemas
		Conceptos de diseño fundamentales
	Conceptos Fundamentales de Programación	Sintaxis básica y la semántica de un lenguaje de alto nivel
		Variables y tipos de datos primitivos
		Expresiones y asignaciones
		Operaciones de E / S
		Archivos de E / S
		Estructuras de control condicionales e iterativas
		Funciones y pasaje de parámetros
		El concepto de recursividad
	Estructuras de datos fundamentales	Listas
		Cadenas y procesamiento de cadenas
		Tipos abstractos de datos y manejo dinámico de memoria
		Referencias y aliasing
		Las listas enlazadas
		Estrategias para la elección de la estructura de datos apropiada
		Búsquedas
		Ordenamiento
	Métodos de Desarrollo	Comprensión Programas
		Corrección del Programas
		Refactorización simple
		Entornos de programación modernos
		Estrategias de depuración
		Documentación y reglas de estilo
Didáctica	Formato clases	¿Se dictan clases netamente teóricas?
		¿Se dictan clases prácticas?
	Ejercitación	¿Se realizan ejercicios de manera colaborativa?
		¿Se realizan ejercicios desde cero?
		¿Se realizan ejercicios a completar?
	Modo Enseñanza	¿Se enseña desde el ejemplo?
		¿Se enseña guiado por el problema?
Herramientas	Lenguaje	¿En qué paradigma se programa?
		¿En qué lenguaje se programa?
		¿En qué entorno se programa?
	Algoritmia	¿Cual es el entorno?
		¿Que leguaje utiliza?
	Campus virtual	¿Cuenta con campus la materia?
		¿El campus se limita a compartir material?
		¿Brinda el campus herramientas de autoevaluación?

3.1. Contenidos

Los aspectos referidos a contenidos los hemos tomado del trabajo propuesto por la "Association for Computing Machinery" (ACM) en conjunto con el "Institute of Electrical and Electronics Engineers" (IEEE) [ACM/IEEE-CS Joint Task Force on Computing Curricula, 2013] por considerarlo un referente internacional para los currículos de computación. En él se sugieren los contenidos y la distribución de los mismos en cursos, estructurados por área de conocimiento y nivel de enseñanza. En el apartado Software Development Fundamentals se centra en el proceso de desarrollo de software, la identificación de los conceptos y las habilidades necesarias que debe abarcar en un curso de programación inicial, los conceptos se subdividen en: algoritmos y diseño, conceptos fundamentales de programación, estructuras de datos fundamentales y métodos de desarrollo.

3.2. Didáctica

Los aspectos referidos a didáctica se elaboraron partiendo de la premisa que para poder utilizar las computadoras en la resolución de problemas de manera efectiva, los estudiantes no solo deben alcanzar las competencias en la lectura y escritura de programas sino, también en el análisis y diseño de soluciones. Nos pareció pertinente determinar el formato de las clases, el modo de ejercitación y si la estrategia de enseñanza responde o no a una metodología tradicional (un modelo por imitación, en donde el instructor propone un problema y desarrolla e implementa su solución, esperando que el estudiante lleve este desarrollo a su propio contexto).

3.3. Herramientas

Los aspectos referidos a herramientas se elaboraron teniendo en cuenta lo expresado por [Ferreira y Rojo, 2006] en relación a que no habria un consenso para enseñar programación, hay métodos de enseñanza que se fundamentan a partir de un paradigma de

programación en particular (funcional, imperativo, imperativo con el aporte de la teoría de objetos) y dentro de ese paradigma se utilizan varios enfoques: enseñar a programar sobre un lenguaje de programación en particular o emplear un lenguaje algorítmico general. En lo referido al lenguaje no solo nos pareció relevante determinar cual es el utilizado y dentro de que paradigma sino también sobre que herramienta se trabaja. Con respecto al uso del campus virtual se pretende determinar, por un lado su utilización y, por otro su alcance, esto es, si se utiliza como soporte de un curso a distancia o como apoyo a la presencialidad.

4. Prueba de Concepto

En esta sección se da una breve descripción de las instituciones consideradas para la prueba de concepto (sección 4.1), y se muestran los resultados y la posible interpretación de los mismos (sección 4.2).

4.1. Instituciones que Componen el Estudio

En esta sección se proporciona una breve descripción de los cursos iniciales de algoritmica de las siguientes instituciones: Creighton University (sección 4.1.1), Grinnell College (sección 4.1.2), Harvard University (sección 4.1.3), Harvey Mudd College (sección 4.1.4), Massachusetts Institute of Technology (sección 4.1.5), Portland Community College (sección 4.1.6), Stanford University (sección 4.1.7), y Worcester Polytechnic Institute (sección 4.1.8).

4.1.1. Creighton University [CU, 2016]
Introduction to Programming, es el primer curso en la secuencia de materias de programación, ofrece una introducción a la resolución de problemas mediante el empleo de la programación. El curso inicia enseñando conceptos de algoritmia y resolución de problemas utilizando el lenguaje gráfico Scratch. Una vez impartidos los primeros conceptos se comienza a utilizar el lenguaje Python, inicialmente empleando el paradigma imperativo para luego brindar los fundamentos del paradigma orientado a objetos. El curso se dicta dos veces por semana durante dos horas, ambas jornadas en laboratorio, integrando teoría con práctica. Los estudiantes completan de seis a ocho tareas, que implican el diseño e implementación de programas en Python, las cuales también pueden incluir en algunos casos un informe escrito en el que se analiza el comportamiento del programa.

4.1.2. Grinnell College [GC, 2016]
Functional problem solving, es el primero de tres cursos existentes en la carrera, cada uno de ellos busca introducir al estudiante a un nuevo paradigma de programación comenzando con el paradigma funcional.
El curso explora mecanismos de representación y manipulación de imágenes. Es prácticamente nula la utilización de algoritmos ya que la materia sólo cuenta con una breve explicación referida a cubrir temas basados en recursividad. El paradigma empleado es el funcional y se utiliza como lenguaje Scheme.
Todas las clases son dictadas en el laboratorio, se enseña en un formato de taller colaborativo, los estudiantes trabajan de a pares en la resolución de ejercicios y se busca que estas parejas cambien semana a semana. Las clases pueden comenzar o terminar con una exposición teórica abierta al debate pero se busca por sobre todo aprovechar el tiempo de clases para trabajar en las computadoras. La materia cuenta con cuatro clases semanales de cincuenta minutos cada una.

4.1.3. Harvard University [HU, 2016]
Introduction to the intellectual enterprises of computer science and the art of programming, también conocido como CS50 enseña a los estudiantes a pensar algorítmicamente y resolver problemas de manera eficiente. Un rasgo a destacar del curso es la cantidad de lenguajes utilizados, se incluyen C, PHP y JavaScript, SQL, CSS y HTML. Una vez impartidos los conocimientos básicos de algoritmia utilizando Scratch se comienza a trabajar utilizando el paradigma imperativo con el lenguaje C, se utiliza este en el periodo

que transcurre desde la semana dos y la semana siete, luego se imparten conceptos referidos a la programación web utilizando como lenguaje PHP y por último JavaScript.
Se encuentra conformado por dos clases teóricas de una hora cada una por semana, es importante destacar que todas las conferencias son transmitidas en vivo y están disponibles en línea desde el momento en que terminan. Adicionalmente los estudiantes cuentan con un espacio denominado "Sections" de noventa minutos semanales, allí los colaboradores de la materia arman grupos reducidos de estudiantes a fin de poder ayudarlos con los temas abordados de esa semana. Por último existe un espacio dos veces por semana de cuatro horas donde los estudiantes pueden asistir a trabajar con las guías prácticas y ser asistidos por los ayudantes. La materia cuenta con nueve guías de ejercicios cada una de ellas en dos modalidades de dificultad.

4.1.4. Harvey Mudd College [HMC, 2016]
Introduction to Computer Science, todos los estudiantes de primer semestre en Harvey Mudd College toman este curso, el mismo imparte los conocimientos elementales de algoritmia en un lenguaje de programación simple denominado Picobot, luego se comienza a trabajar con Python, en un enfoque multiparadigma el cual se denomina "Breadth-First". Este es un curso que se encuentra conformado por dos conferencias de setenta y cinco minutos por semana y un laboratorio opcional, pero que atrae a más del 90% de los estudiantes a una sesión suplementaria de dos horas a la semana. Las tareas contienen uno o más problemas "individuales" que cada estudiante debe completar por su cuenta, el resto de los problemas, es posible completarlos junto a otro estudiante.

4.1.5. Massachusetts Institute of Technology [MIT, 2016]
Introduction to Computer Science and Programming, tiene por objetivo brindar las herramientas necesarias para permitir que el estudiante pueda resolver problemas empleando la programación. No se utilizan algoritmos, el curso comienza mostrando a los estudiantes el entorno de programación, presentando algunas sentencias y haciendo que desde el inicio se realicen pequeños scripts en Python. Al tratarse de Python el lenguaje de programación elegido permite comenzar a utilizarlo de manera imperativa para pasar poco a poco al paradigma de objetos. Existen dos clases semanales de una hora cada una, adicionalmente los estudiantes cuentan con una clase de una hora pensada para resolver dudas, allí los estudiantes tienen la oportunidad de hacer preguntas sobre el material de lectura o sobre el problema planteado para la semana. La materia cuenta con una guía de ejercicios que apunta a que cada estudiante la resuelva de manera individual, muchos de estos ejercicios cuentan con "archivos de soporte" en ellos se encuentra el esqueleto del problema, en muchos casos con una exhaustiva explicación asociada a cada función, de manera que puede considerarse como una actividad de práctica semicontrolada ya que el estudiante debe ceñirse al esqueleto del programa que recibe.

4.1.6. Portland Community College [PCC, 2016]
Java Programming I, se enseñan conceptos de diseño, implementación y pruebas de software utilizando Java, utilizando un entorno de desarrollo diseñado para la enseñanza a un nivel introductorio, denominado BlueJ, este permite comenzar a interactuar con el concepto de objetos de manera gráfica. El paradigma empleado es de programación orientada a objetos y el lenguaje utilizado es Java. El enfoque del curso es el de first objects (objetos primero) con el fin de concentrarse más en los conceptos y menos en los detalles sintácticos y prácticos necesarios para conseguir un programa en ejecución. El curso está formado por cuarenta horas de clase teórica-prácticas y veinte horas de laboratorio opcionales. Todas las aulas están equipadas con una computadora en cada escritorio y el tiempo de clase se compone de dos conferencias y luego actividades en equipos. Los estudiantes suelen trabajar en los ejercicios y hacer preguntas relacionada con estos. A lo largo del curso se trabajan siete

proyectos de programación, en seis de ellos los estudiantes añaden funcionalidad al código existente, que van desde la adición de un único método de una sola línea hasta añadir funcionalidad significativa y sólo en uno de los proyecto los estudiantes escriben todo el código desde cero.

4.1.7. Stanford University [SU, 2016]
Programming Methodology, es un curso que fue diseñado para estudiantes de múltiples especialidades y no requiere conocimientos previos en programación.
El curso comienza utilizando Java con Karel "el Robot", una herramienta de aprendizaje que presenta los conceptos de una forma visual. Una vez incorporados los conceptos básicos de programación mediante Karel, el lenguaje utilizado es Java y en él se trabajan los aspectos formales de programación utilizando el paradigma de orientación a objetos. El curso se dicta tres veces por semana durante una hora y el formato es netamente teórico, no cuenta con prácticas de laboratorio. Los estudiantes pueden solicitar entrevistas con docentes de la clase que brindan apoyo a fin de poder evacuar dudas sobre los aspectos prácticos de la materia. A lo largo de la cursada existen siete trabajos prácticos los cuales son evaluados y sobre ellos es posible obtener una explicación detallada de los errores cometidos.

4.1.8. Worcester Polytechnic Institute [SU, 2016]
Introduction to Program Design, es un curso para estudiantes sin experiencia previa en programación, el curso está pensado para aprender a diseñar programas que resuelvan problemas. Prácticamente es nula la utilización de algoritmos, el curso introduce a la disciplina de la informática centrándose en la resolución de problemas desde la perspectiva de la programación funcional. El lenguaje utilizado en el curso es Racket, un lenguaje de programación de amplio espectro de la familia de Lisp y Scheme. El curso cuenta con cuatro horas semanales de clases teóricas prácticas y una clase de una hora pensada para resolver un ejercicio en el laboratorio. Se enfatiza en la metodología de desarrollo conocida como data-driven. Los estudiantes trabajan en una tarea de programación corta durante la clase de laboratorio cuya duración es de una hora y les es asignada una guía de ejercicios relacionada con los temas de programación abordados en las clases teóricas.

4.2. Resultados e Interpretación

Los resultados relevados en el protocolo propuesto se presentan el la Tabla 2. Estos resultados permiten formular las siguientes conclusiones preliminares:

- Solo en la mitad de los casos relevados se utilizan algoritmos y que en los casos donde se aplican se ha optado por utilizar una herramienta software que de soporte.
- Notamos un bajo índice de cobertura en temas referidos a operaciones de entrada salida, manejo de archivos, listas enlazadas y referencias.
- Es prácticamente nula la cobertura en el tema refactorización simple, en el único caso donde se lo implementa se realizan practicas donde los estudiantes deben realizar correcciones mínimas al código de los programas para permitir que estos logren una funcionalidad determinada.

  En lo referido a metodologías de ejercitación a pesar que la tendencia preponderante es la práctica desde cero se detectaron tres casos en los cuales no se utiliza como única metodología de ejercitación, en estos casos se alterna con ejercicios a completar por parte de los estudiantes. Se destaca entre los que optaron por incorporar otra metodología de ejercitación, en el [PPC, 2016] a lo largo de la cursada se desarrollan siete proyectos de programación, en seis de los cuales los estudiantes añaden funcionalidad al código existente y tan solo en el último proyecto los estudiantes escriben todo el código desde cero. Otro de los casos en los que se promueven las prácticas semi-libres es en el [MIT, 2016] allí los estudiantes cuentan con una guía de ejercicios con "archivos de soporte" y en ellos se encuentra el esqueleto del pro-

blema, en muchos casos con una exhaustiva explicación asociada a cada función.

Tabla 2. Resultados del Relevamiento de Información con el Protocolo Propuesto

			CUO	GC	Harvard	HMC	MIT	PCC	Stanford	WPI
Contenido	Algoritmos y Diseño	El concepto y propiedades de los algoritmos	SI	NO	SI	SI	NO	NO	SI	NO
		El papel de los algoritmos en el proceso de resolución de problemas	SI	NO	SI	SI	NO	NO	SI	NO
		Las estrategias de resolución de problemas	SI	SI	SI	SI	SI	SI	SI	SI
		Conceptos de diseño fundamentales	SI	SI	SI	SI	SI	SI	SI	SI
	Conceptos Fundamentales de Programación	Sintaxis básica y la semántica de un lenguaje de alto nivel	SI	SI	SI	SI	SI	SI	SI	SI
		Variables y tipos de datos primitivos	SI	SI	SI	SI	SI	SI	SI	SI
		Expresiones y asignaciones	SI	SI	SI	SI	SI	SI	SI	SI
		Operaciones de E / S	SI	NO	SI	NO	SI	SI	SI	NO
		Archivos de E / S	SI	NO	SI	NO	NO	SI	SI	NO
		Estructuras de control condicionales e iterativas	SI	SI	SI	SI	SI	SI	SI	SI
		Funciones y pasaje de parámetros	SI	SI	SI	SI	SI	SI	SI	SI
		El concepto de recursividad	SI	SI	SI	SI	SI	SI	NO	SI
	Estructuras de datos fundamentales	Listas	SI	SI	SI	SI	SI	SI	SI	SI
		Cadenas y procesamiento de cadenas	SI	SI	SI	SI	SI	SI	SI	NO
		Tipos abstractos de datos y manejo dinámico de memoria	SI	NO	SI	SI	SI	SI	SI	NO
		Referencias y aliasing	NO	NO	SI	NO	SI	SI	SI	NO
		Las listas enlazadas	NO	NO	SI	NO	NO	SI	NO	SI
		Estrategias para la elección de la estructura de datos apropiada	SI	SI	SI	SI	SI	SI	SI	NO
		Búsquedas	SI	SI	SI	SI	SI	SI	SI	NO
		Ordenamiento	NO	SI	SI	SI	SI	SI	SI	NO
	Métodos de Desarrollo	Comprensión Programas	SI	SI	SI	SI	SI	SI	SI	SI
		Corrección del Programas	SI	SI	SI	SI	SI	SI	SI	SI
		Refactorización simple	NO	NO	NO	NO	NO	SI	NO	NO
		Entornos de programación modernos	SI	NO	SI	SI	SI	SI	SI	NO
		Estrategias de depuración	SI	SI	SI	SI	SI	SI	SI	SI
		Documentación y reglas de estilo	SI	SI	SI	NO	SI	SI	SI	NO
Didáctica	Formato clases	¿Se dictan clases netamente teóricas?	NO	NO	SI	SI	SI	NO	SI	SI
		¿Se dictan clases prácticas?	SI	SI	SI	SI	NO	SI	SI	SI
	Ejercitación	¿Se realizan ejercicios de manera colaborativa?	NO	SI	SI	NO	NO	NO	NO	SI
		¿Se realizan ejercicios desde cero?	SI	SI	SI	SI	SI	SI	SI	SI
		¿Se realizan ejercicios a completar?	NO	NO	SI	NO	SI	SI	NO	NO
	Modo Enseñanza	¿Se enseña desde el ejemplo?	SI	SI	SI	SI	SI	SI	SI	SI
		¿Se enseña guiados por el problema?	SI	NO	NO	NO	NO	NO	NO	NO
Herramientas	Lenguaje	¿En qué paradigma se programa?	Mixto	Funcional	Imperativo	Mixto	Mixto	OO	OO	Funcional
		¿En qué lenguaje se programa?	Python	Scheme	C/PHP/JS	Python	Python	Java	Java	Racket
		¿En qué entorno se programa?	IDLE	Gimp	CS50 IDE	IDLE	IDLE	Blue J	Eclipse	Dr Racket
	Algoritmia	¿Cual es el entorno?	Scratch	-	Scratch	Picobot	-	-	Karel	-
		¿Que leguaje utiliza?	Scratch	-	Scratch	Picobot	-	-	Java	-
	Campus virtual	¿Cuenta con campus la materia?	SI	SI	SI	SI	SI	SI	SI	SI
		¿El campus se limita a compartir material?	SI	SI	NO	SI	SI	SI	SI	SI
		¿Brinda el campus herramientas de autoevaluación?	NO	NO	SI	NO	NO	NO	NO	NO

- Se detecto que impera el modelo en el cual se dictan clases netamente teóricas para luego dar lugar a las prácticas en el laboratorio, tan solo en tres de los casos se trabaja íntegramente con clases teórico prácticas quedando la clase de consulta o práctica con asistencia opcional. El modo predominante a la hora de impartir conocimiento es desde el ejemplo, mediante éste método el docente explica el marco teórico y refuerza con ejemplos prácticos para luego permitir ejercitar a los estudiantes.
- Todos los casos analizados utilizan campus virtual, en el cómo se lo utiliza se detectaron tres:

[i] Los casos que se limitan solo a compartir información, poner a disposición ejercicios y material de lectura.

[ii] Los casos del Massachusetts Institute of Technology y el de Stanford University, en ambas casas de estudio notamos que se busca extender el aula y permitirle al estudiante acceder no solo al material teórico sino también a las grabaciones de las clases.

[iii] El caso de Harvard University, se incorpora a los videos de las clases teóricas tres agregados:

- *Streaming*, todas las clases teóricas de Harvard y Yale son transmitidas en vivo y luego quedan a disposición de los estudiantes .
- *Walkthroughs*, vídeos a través de los cuales los colaboradores del curso indican a los estudiantes por dónde empezar y cómo abordar un desafío. Se espera que el estudiante vea estos tutoriales antes de hacer preguntas sobre el problema en horario de oficina o por medio del foro.
- *Postmortems*, disponibles después de las fechas límite de entrega de las guías de ejercicio, a través de estos videos los responsables del curso muestran soluciones reales a los problemas planteados en la guía.

- Se detectó que existe una marcada tendencia en la utilización del paradigma orientado a objetos, ya sea en los dos casos donde se utiliza a éste como único paradigma de la cursada como en aquellos casos donde se trabaja con múltiples paradigmas y en todos los casos el orientado a objetos forma parte. Es importante destacar que en todos los casos donde se opta por trabajar con un enfoque mixto el lenguaje empleado es Python.
- En líneas generales se opta por trabajar con entornos de programación modernos, el caso que lleva esto un paso mas allá es el de Harvard con el "CS50 IDE", ya que no solo se trata de un entorno moderno sino que es un entorno de desarrollo integrado alojado en la nube por lo tanto que solo requiere tener un navegador web.

5. Conclusiones

La motivación para proponer el protocolo surge de la necesidad de disponer de un instrumento que nos permita analizar uno de los posibles vectores de deserción en los cursos de programación inicial.

Si bien consideramos que el instrumento se encuentra estabilizado y que los resultados alcanzados pueden considerarse como satisfactorios, no descartamos, a futuro, realizar ajustes sobre el mismo.

Como trabajo futuro planteamos la elaboración de un cuestionario a ser enviado a los responsables de las asignaturas de programación inicial de Carreras de Informática a nivel nacional a efectos de replicar la experiencia con información de nuestro sistema universitario.

6. Financiamiento

Las investigaciones que se reportan en este artículo han sido financiadas parcialmente por el Proyecto de Investigación 33B180 de la Universidad Nacional de Lanús.

7. Referencias

Barg, M., Fekete, A., Greening, T., Hollands, O., Kay, J., Kingston, J. H. & Crawford, K., *Problem based learning for foundation computer science courses*. Computer Science Education, 2000

Bayman, P. and Mayer, R. E. 1983. *A diagnosis of beginning programmers' misconceptions of BASIC programming statements*. Commun. ACM 26, 9 (Sep. 1983), 677-679.

Becker, B. W. 2001. *Teaching CS1 with karel the robot in Java*. ACM SIGCSE Bulletin Vol. 33, No. 1, pp. 50-54.

Brito, M., & de Sá-Soares, F. 2014. *Assessment frequency in introductory computer programming disciplines*. Computers in Human Behavior, 30, 623-628.

Byrne, P., & Lyons, G. 2001. *The effect of student attributes on success in*

programming. In ACM SIGCSE Bulletin (Vol. 33, No. 3, pp. 49-52). ACM.

CU, 2016. *Introduction to Programming*. Creighton University. http://dave-reed.com/csc221.F13. Pagina vigente al 28/03/2016

Estácio, B., Oliveira, R., Marczak, S., Kalinowski, M., Garcia, A., Prikladnicki, R., & Lucena, C. 2015 *Evaluating Collaborative Practices in Acquiring Programming Skills: Findings of a Controlled Experiment*.. Proceedings of 29th IEEE Brazilian Symposium on Software Engineering (SBES), 2015 Pág. 150-159.

GC, 2016. *Functional problem solving*. Grinnell College. http:// www.cs.grinnell.edu/~davisjan/csc/151/2013F/ Pagina vigente al 28/03/2016

HU,2016. *Introduction to the intellectual enterprises of computer science and the art of programming*. Harvard University. https://cs50.harvard. edu/ Pagina vigente al 28/03/2016

HMC, 2016. *Introduction to Computer* Science. Harvey Mudd College. https://www.cs.hmc.edu/twiki/bin/view/CS5. Pagina vigente al 28/03/2016

Klassner, F., & Anderson, S. D. 2003. *Lego MindStorms: Not just for K-12 anymore*. IEEE Robotics & Automation Magazine, 10(2), 12-18.

Leone, L., Veizaga, K., Conforte, J., & Zanazzi, J. L. 2014. *Modelos para explicar el desgranamiento en una carrera de Ingeniería*. En Actas de las XLIII Jornadas Argentinas de Informática e Investigación Operativa (43JAIIO)-XII Simposio Argentino de Investigación Operativa (SIO)(Buenos Aires, 2014).

Ma, L., Ferguson , J., Roper, M., and Wood, M. 2007. *Investigating the viability of mental models held by novice programmers*. Proceedings of the Thirty-Eighth SIGCSE Technical Symposium on Computer Science Education. SIGCSE '07.

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. 2006. *Pair programming improves student retention, confidence, and program quality*. Communications of the ACM, 49(8), 90-95.

MIT, 2016. *Introduction to Computer Science and Programming*. Massachusetts Institute of Technology. http://ocw.mit.edu/courses/electrical-engineering-and-computer-science. Pagina vigente al 28/03/2016.

PCC, 2016. *Java Programming I*. Portland Community College. http://www.pcc.edu/ccog/default.cfm?fa=ccog&subject=CIS&course=13. Pagina vigente al 28/03/2016

Porter, L., & Simon, B. 2013. *Retaining nearly one-third more majors with a trio of instructional best practices in CS1*. Proceeding of the 44th ACM technical symposium on Computer science education. Pág. 165-170).

Porter, L., Bailey Lee, C., Simon, B., Cutts, Q., & Zingaro, D. 2011. *Experience report: a multi-classroom report on the value of peer instruction*. Proceedings of the 16th ACM Annual Joint Conference on Innovation and Technology in Computer Science Education. Pág. 138-142.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., & Kafai, Y. 2009. *Scratch: programming for all*. Communications of the ACM, 52(11): 60-67.

Rimington, K. B. 2010. *Expanding the Horizons of Educational Pair Programming: A Methodological Review of Pair Programming in Computer Science Education Research*. Master Thesis on on Computer Science. Utah State University.

Rooksby, J., Hunt, J., & Wang, X. 2014. *The theory and practice of randori coding dojos*. En Agile Processes in Software Engineering and Extreme Programming Pág. 251-259. Springer International Publishing.

Simon, B., Kohanfars, M., Lee, J., Tamayo, K., & Cutts, Q. 2010. *Experience report: peer instruction in introductory computing*. Proceedings of the 41st ACM technical symposium on Computer science education Pág. 341-345.

Spohrer , J., & Soloway, E. 1986. Alternatives to construct-based programming

misconceptions. ACM SIGCHI Bulletin, 17(4): 183-191.

SU, 2016. *Programming Methodology*. Stanford University. http://web.stanford.edu/class/cs106a/. Pagina vigente al 28/03/2016

Sung, K., Panitz, M., Wallace, S., Anderson, R., & Nordlinger, J. 2008. *Game-themed programming assignments: the faculty perspective*. ACM SIGCSE Bulletin, 40(1): 300-304).

WPI, 2016. *Introduction to Program Design*. Worcester Polytechnic Institute. http://web.cs.wpi.edu/~cs1101/a15/. Pagina vigente al 28/03/2016