# Criterios de Análisis de Software Educativo para Aprender a Programar

Lucas Spigariol
*Universidad Tecnológica Nacional (FRD – FRBA)*
*Universidad Nacional de San Martín*
*lspigariol@gmail.com*

## Abstract

*Desde una lectura del contexto actual de la actividad profesional en el campo de los Sistemas de Información y de la dinámica educativa, se presentan diferentes criterios de evaluación para analizar la pertinencia de utilizar determinados productos de software educativo como herramienta didáctica para enseñar a programar en el ambiente universitario, en vistas a formar profesionales en informática. Luego, se hace un recorrido por diferentes casos de herramientas desarrollas con un fin pedagógico y se focaliza el análisis en dos casos particulares, aplicando sobre ellas los mencionados criterios de evaluación. Se trata de la plataforma web Mumuki y del lenguaje de objetos Wollok, dos herramientas diferentes en cuanto a sus objetivos pero que confluyen en aspectos técnicos de su implementación, que articulan sus equipos de desarrollo, y que comparten buena parte de la comunidad educativa que los utiliza en la actualidad en diversas universidades e instituciones educativas, entre las que se destaca la Universidad Tecnológica Nacional, la Universidad Nacional de Quilmes y la Universidad Nacional de San Martín.*

## 1. Introducción

Si bien en la realidad se presentan inter-relacionadamente, en una aproximación analítica se identifican aspectos específicos a considerar a la hora de evaluar el uso de una determinada herramienta para ser utilizada como recurso pedagógico en el marco asignaturas que tienen por objetivo que el estudiante aprenda a programar. Podrían agregarse otras, podrían ser menos, podrían agruparse de otra manera o subdividirse, pero en el presente trabajo se hace foco en seis características importantes que ameritan identificarse. Sin dudas que no se pretende separarlas quirúrgicamente entre ellas ni ignorar sus mutuas dependencias, tampoco su orden implica una ponderación o jerarquización en particular, pero como modo de organizar su presentación, se las va recorriendo secuencialmente.

Sin perder de vista el carácter técnico del desarrollo de software que está emparentado con las ciencias exactas, las comúnmente denominadas "duras", este trabajo se enmarca en una perspectiva pedagógica, con una metodología más propia de las ciencias humanas. Se hablará de programas, de código, de sistemas, de polimorfismo y otros términos técnicos, pero más precisamente, de la complejidad del proceso de enseñanza y aprendizaje de estos conceptos. En ese sentido, se asume un método dialectico de construcción del conocimiento, por el que los criterios están planteados a modo de tensión: la argumentación que fundamente a cada uno de ellos discursivamente discute con otros posicionamientos a quienes no se pretende identificar con nombre y apellido, pero que remiten a ideas o prácticas que -algunas más explícitas, otras latentes- están presentes en el ambiente universitario y profesional.

- ***Conceptos antes que lenguajes***
- ***Más que programar, construir software***
- ***Lo profesional como punto de llegada, no de partida.***
- ***Lo visual, interactivo y lúdico no es sólo es una cuestión de niños.***
- ***Herramientas apropiadas, sin enlatar.***

### 1.1. Conceptos antes que lenguajes

Para comenzar, un ejemplo. Aprender cómo se pasa un parámetro o cómo se lo recibe en algún lenguaje de programación es algo relativamente sencillo: será utilizando paréntesis o separando con comas, será con dos puntos, una llave u otro carácter, habrá o no que declarar el tipo de dato, tal vez haya que anteponer alguna palabra reservada para indicar que se trata de una referencia o no haga falta, puede que el lenguaje tenga más de una sintaxis alternativa para hacer lo mismo… hay variantes y particularidades que quien quiera desarrollar software

utilizando el lenguaje en cuestión seguramente deba saber. Ahora bien, lo más importante que esta persona debe saber es *qué delegar* y a *quién*. Sin tener claro el concepto de *delegación*, saber cómo codificar el pasaje de un parámetro sirve de poco. Podríamos añadir que esta persona también, dependiendo del caso, deba saber también con qué criterio *modularizar* el programa, qué consecuencias trae definir esa *interfaz* de cierta manera, tener en cuenta si con ese parámetro mantiene el *polimorfismo*, etc, pero para el ejemplo basta con limitarse a la idea de *delegación*. Saber un concepto es mucho más que saber cómo se lo escribe en un lenguaje, es saber el porqué y el para qué, poder analizar ventajas y desventajas, es comprender el problema y poder justificar cuándo es conveniente la aplicación de dicho concepto para resolverlo.

Muchas veces, los cursos de programación se focalizan más en lenguajes y no prestan detenida atención a los conceptos que les dan sentido, en el mejor de los casos asumiendo -y advirtiendo- que se trata de una propuesta pedagógica destinada a personas que saben programar y que quieren aprender la forma de hacerlo en un nuevo lenguaje, aunque otras veces ofreciendo al alumnado una propuesta de baja calidad, sin favorecer buenas prácticas de programación y en definitiva desaprovechando la potencialidad de las herramientas de desarrollo de software.

De esta manera, un criterio central para analizar una herramienta es ver en qué medida favorece la comprensión de conceptos de programación y orienta al estudiante para ponerlos en práctica de forma que sea evidente su sentido y utilidad para construir soluciones. Más aún, también es ver qué conceptos vale la pena enseñar con mayor dedicación, con qué enfoque abordar desde un conocimiento actualizado del ejercicio profesional. Por ejemplo, la lectura de un archivo secuencial de registros, más allá de la utilidad que se le pueda encontrar al concepto para un problema fabricado por los docentes, cabe preguntarse cuánta aplicación tiene en un ambiente profesional marcado por bases de datos con otras formas de acceso más sofisticadas.

Por cierto, depende mucho también de la formación del docente y de la manera en que prepara y lleva adelante su clase. Si tampoco tiene claro la utilidad del concepto o llega al aula con su *cassette* que repite año tras año mecánicamente, ninguna herramienta le puede ayudar; en cambio, si tiene claro su objetivo pedagógico y se toma la molestia de preparar su clase, podrá ver que hay ciertas herramientas que le resultan más adecuadas que otras para priorizar la enseñanza de conceptos de programación.

En el otro extremo, hay quienes afirmándose en la importancia del concepto ponen en un segundo plano todo lenguaje de programación y se limitan a un planteo teórico que no se *contamina* con ninguna implementación. Desde la concepción que fundamente este trabajo, la práctica es fundamental y su articulación con la teoría es imprescindible para precisamente comprobar las ventajas o desventajas concretas de los conceptos y cómo impactan en una solución. En este sentido, retomando uno de los principios básicos de la pedagogía crítica latinoamericana, es preciso "entender la práctica docente como un quehacer crítico, creador y recreador; asumir la relación conflictiva y contradictoria entre teoría y práctica, no como una dicotomía, sino como interrelación fecunda que da lugar a nuevos aprendizajes". [1] Y para ello, tratándose de conceptos de desarrollo de software, requiere inexorablemente la utilización de algún lenguaje, o más genéricamente, de una herramienta de desarrollo de software.

## 1.2. Más que programar, construir software

Quienes forman parte activa de la comunidad universitaria, como en toda institución, saben de sus puntos débiles y sus aciertos, en otras palabras, le conocen las mañas. Por un lado, es la cuna del saber, heredera de una larga tradición de producción de conocimiento, de investigación y desarrollo, la vanguardia de la intelectualidad, la cresta de la ola de la innovación tecnológica… Por otra parte, es un lugar seguro para trabajar, donde mientras se respeten ciertas normas mínimas, está garantizado el puesto hasta la jubilación, sin importar tanto si los contenidos están actualizados, si la metodología es apropiada, si el perfil del alumnado cambió, etc… en esto la libertad de cátedra puede ser un arma de doble filo. No se trata de cuantificar ambos extremos ni ignorar el abanico de posiciones intermedias que se pueden encontrar. Viene a colación de lo siguiente: En materia de sistemas de información, el cambio se da de una manera vertiginosa que es difícil de seguir por las universidades. Concretamente, hay cierta inercia de seguir pensando que la programación es como era hace unas décadas atrás, o incluso reconociendo la existencia de nuevos lenguajes de programación, que los conceptos que hay detrás y la forma de trabajo siguen siendo los mismos.

Resulta oportuno rescatar los aportes de Paulo Freire respecto de la actitud docente de *leer el mundo* donde recomienda tomar la distancia necesaria para leer el mundo, "entendiéndolo como el contexto en el cual se logra una comprensión más exacta del objeto de estudio", invita a "revisar permanentemente que los contenidos sean saberes socialmente relevantes, de carácter histórico y en relación con conocimientos académicos" y sugiere "abandonar las interpretaciones mecanicistas o idealistas de la realidad, asumiendo que las relaciones entre la conciencia y el mundo son dialécticas". [2]

Pero más que a los conceptos de programación en sí, que como se señaló anteriormente van cambiando, hay también un cambio grande en la dinámica misma de

desarrollo de software. Precisamente, no es casual el pasaje del término "programar" por el de "construir software" en la frase anterior.

Existe una cierta concepción que limita la tarea de programar al uso de un lenguaje de programación. En otras palabras, que asume al software como un simple texto y a su proceso de desarrollo como una tarea de escritura. Desde este enfoque, el ambiente de desarrollo es cualquier editor de texto -fácilmente sustituible por el pizarrón en el aula o por una hoja de papel en un examen- y que en algunos casos viene complementado con el compilador adecuado que transforma ese código en un programa ejecutable. La dinámica de trabajo intercala la compilación, la ejecución y nuevamente la edición, de acuerdo a lo que se observe en cada iteración como resultado o como error.

En cambio, la dinámica actual de desarrollo de software tiende a utilizar *entornos integrados de desarrollo* (IDE) que no sólo permiten escribir código, sino que incluyen diversas herramientas que facilitan el trabajo de programar y se integran con el compilador o intérprete correspondiente. Entre otros tantos cambios, se utilizan *tests unitarios* como parte del proceso de desarrollo, el código se comparte y versiona mediante repositorios, se le presta una mayor atención a la arquitectura que sustenta se combinan diferentes herramientas para crear y articular los diferentes componentes (bases de datos, interfaz de usuario, conectividad, servicios externos, etc.).

Por otra parte, como metodología de trabajo, se asume un vínculo más fluido entre el análisis, diseño, implementación y prueba, reemplazando la linealidad de otras épocas por un esquema espiralado que incorpora complejidad progresivamente.

## 1.3. Lo profesional como punto de llegada, no de partida

No por poco novedoso deja de ser fundamental tener presente la progresividad como criterio transversal de la tarea educativa. En un marco donde teoría y práctica se articulan y enriquecen mutuamente, una preocupación permanente del docente es cómo ir abriendo un camino que vaya incrementando paulatinamente la complejidad de los conceptos, en los que unos dan fundamento a otros y facilitan el proceso de aprendizaje de los estudiantes.

Las herramientas profesionales de desarrollo de software están pensadas para que quien domina con fluidez los conceptos de programación, pueda hacerlos confluir en un caso concreto y maximizar el rendimiento de su trabajo. En contrapartida, para un estudiante inicial de programación, encontrarse de pronto con una cantidad enorme de conceptos y herramientas, constituye un problema.

Por ejemplo, en un lenguaje como *Java*, que para escribir un simple *hola mundo* tenga que definir un objeto a partir de una *clase*, que deba ubicar esa clase dentro de una *jerarquía* con *herencia*, que deba definir un *método* y declararlo utilizando ciertas palabras reservadas y a eso sumarle un uso adecuado de la sintaxis, representa una cantidad enorme de conceptos, que al menos abre a la reflexión acerca de si es la herramienta más adecuada para enseñar a programar en objetos.

Se propone analizar las posibilidades que una herramienta ofrece para una dinámica de ocultamiento/develamiento, en la que se oculta inicialmente cierta complejidad conceptual en vistas a un oportuno develamiento. Esta estrategia llevada adelante mediante mediaciones para procurar una curva suave de aprendizaje, remite a lo que Vigotsky propone con el concepto de *zona de desarrollo próximo*. [3] En otras oportunidades, la herramienta puede tener la función de un *tutor*, que en determinado momento del *crecimiento* cumple un rol de sostén y que luego es prescindible, como lo sugiere el concepto de *andamiaje*. [4]

Ninguna herramienta de por sí puede garantizar esto, pero puede presentar características que le permitan -o no- al docente que las inserte en su propuesta pedagógica, la articule con su forma de llevar adelante la secuencia didáctica, y gradúa su aplicación.

## 1.4. Lo visual, interactivo y lúdico no es sólo es una cuestión de niños

Se reconoce la importancia de todo esfuerzo para lograr que el estudiante no sólo no le represente un obstáculo tener que utilizar una herramienta que no entiende, que le resulta antigua por su aspecto o que es estática, sino que le sea sencillo usarla, que favorezca su motivación, que despierte o abone su interés por la programación y las ciencias de la computación en general.

El juego, el carácter lúdico de una propuesta educativa, lejos de ser un recurso exclusivo para niños, también tiene sentido en un ambiente de formación profesional o universitario, en la medida en que se lo administra criteriosamente, no como mero espectáculo sino con un fin pedagógico determinado, sin abusar de él para no infantilizar al alumnado.

Todo lo gráfico y visual, también aplicado con sentido explicativo y no como un simple decorado, aporta un valor adicional para la comprensión.

De esta manera, enmarcado dentro de una propuesta curricular que lo contiene, el uso de dramatizaciones, desafíos, metáforas, humor o juegos son recursos que, así como resultan valioso para una dinámica áulica, también pueden estar presente en una herramienta informática con fines educativos.

### 1.5. Herramientas apropiadas, sin *enlatar*

Lejos de interpretarlo desde uno de sus posibles significados del término *apropiación* en tanto robo, se plantea el concepto de una herramienta *apropiada* en un doble sentido: Por un lado, como sinónimo de adecuada, propicia, como una herramienta que se adapta y acomoda a una propuesta pedagógica como recurso para enseñar a programar. Por otra parte, sugiere una actitud de parte del docente de *hacer propia* a una herramienta, de no hacer un uso mecánico o superficial, sino de sentirse dueño de ella, de asumirla como parte de su propuesta educativa, de configurarla, y por qué no modificarla. En otras palabras, hablar de *herramienta* también tiene un sentido que remite más a un artesano que a un maquinista, que asocia la tarea educativa a la de un artista antes que a la de un empleado.

Si bien en toda disciplina es posible encontrar software educativo que ayude al proceso pedagógico, en el área informática se da una coincidencia no menor: es un ámbito donde la necesidad y la capacidad confluyen. Es posible hacer programas para enseñar a programar. Es factible desarrollar software que se utilice con fines educativos en el mismo ámbito en que se los construye. En otras palabras, que docentes de programación utilicen como recurso pedagógico software desarrollado por ellos mismos, es una fecunda particularidad que abre enormes posibilidades. En concreto, existen numerosos grupos de docentes e instituciones educativas que a raíz de su preocupación pedagógica acerca del aprendizaje de la programación, hacen su propia lectura del panorama profesional del desarrollo de software y la realidad de sus estudiantes y a partir de sus reflexiones conclusiones seleccionan, desarrollan, adaptan o de alguna otra manera se apropian de herramientas de software específicas con fines educativos.

En el mundo de sistemas, el primer umbral que habilita a la apropiación de una herramienta de software por parte del usuario -al menos del calificado profesionalmente como se supone que es un docente universitario del área- es que se trate de software libre. Frente al abuso que se hace en ocasiones del software *enlatado*, se defiende la libertad de las herramientas y materiales de estudio como recurso fundamental para la democratización del conocimiento. Esta mirada representa para todos aquellos que asumen el desafío de la educación con un posicionamiento complejo y explícito: la opción por una educación crítica, comprometida y esperanzada, sin por ello perder calidad, todo lo contrario. Porque "educar sigue siendo un desafío que nos convoca éticamente, es una práctica ciudadana y política: cómo educar, por qué a quién, para qué sociedad, pensando en qué presente y en qué futuro, para qué país. Educar se sostiene también hoy – pese a que muchos intenten contradecirnos con argumentaciones falaces-, en los mismos valores democráticos de urbanidad, respeto, solidaridad, ética, justicia y responsabilidad." [5]

## 2. Relevamiento de experiencias

A modo de paneo, se presentan una serie de herramientas para aprender a programar, de diversa índole. Hay algunas más orientadas a educación básica y otra a un perfil universitario, están las realizadas en el país y otras que provienen de otras latitudes, hay plataformas web, lenguajes de programación, bibliotecas, *frameworks* y otras aplicaciones. Sin pretender que el elenco sea exhaustivo, se mencionan algunas de ellas que resultan interesantes.

Siguiendo el camino abierto por *Logo*, el lenguaje educativo emblemático creado décadas atrás en el Instituto Tecnológico de Massachusset (MIT) por Seymour Papert, hay otros lenguajes creados específicamente con un fin pedagógico. Entre ellos se destaca una Scratch, desarrollado también en el MIT. Quienes llevaron adelante la reflexión pedagógica del proyecto que enfatiza el hecho de diseñar, en tanto crear y no solamente interactuar con cosas ya hechas, plantean un modelo de aprendizaje colaborativo y reflexivo, revisando permanentemente las propias prácticas. Afirman que "aprendiendo las lecciones de la experiencia de *Papert* con *Logo*, hemos diseñado *Scratch* para ir más allá de *Logo*, en tres dimensiones, por lo que la programación más manipulable, más significativo, y más social." [6]

Un desarrollo argentino que retoma una de las características más visibles de *Scratch* es un emprendimiento de la Fundación Sadosky denominado *Pilas Bloques* (www.pilasbloques.program.ar). Precisamente, se basa en construir software con una herramienta gráfica que consiste en bloques que se manipulan a modo de rompecabezas, en vez de escribir el código secuencialmente. Viene asociado a una serie de desafíos para resolver que proponen un recorrido pedagógico de incorporación de conceptos y habilidades de programación, con un estilo marcadamente lúdico, orientado a niños y adolescentes.

Con un enfoque muy diferente, otra herramienta es un intérprete de pseudocódigo denominado *PSeInt* (pseint.sourceforge.net), que sirve para enseñar los conceptos básicos y las abstracciones para empezar a programar, sin tener que preocuparse por un lenguaje y sus particularidades. Consta de un editor con coloreado de sintaxis, autocompletado y ayuda rápida en pantalla. Lo más interesante de la herramienta es que permite visualizar el algoritmo como diagrama de flujo y de convertirlo a código en lenguaje C++ o viceversa. La computadora interpreta y ejecuta el algoritmo y hace un seguimiento gráfico de su ejecución mediante el diagrama, mostrando por donde trascurre la ejecución.

Existen varios sitios web que proponen una interactividad con el usuario para resolver ejercicios de programación con diferentes lenguajes y recursos visuales. Entre ellos por ejemplo se encuentra uno llamado *LightBot* (lightbot.com) que es muy intuitivo y simple y está también el emprendimiento *Code* (code.org) que es muy completo y con gran producción visual.

Otro camino lo conforman ciertas bibliotecas o *frameworks* que trabajan sobre lenguajes de programación de uso profesional, pero donde escondiendo su complejidad detrás de una interfaz amigable, a partir de la motivación por crear videojuegos introduce en la programación. *Pygame* (www.pygame.org), *Cocos2d* (www.cocos2d-x.org) *Pilas Engine* (pilas-engine.com.ar) son algunas experiencias en ese sentido.

A las ya señaladas se agregan otras dos, sobre las cuales se centra el análisis que propone el presente trabajo. Ellas son *Mumuki* (www.mumuki.org) y *Wollok* (www.wollok.org), dos herramientas diferentes en cuanto a sus objetivos pero que confluyen en aspectos técnicos de su implementación, que articulan sus equipos de desarrollo, y que comparten buena parte de la comunidad educativa que los utiliza en la actualidad en diversas universidades e instituciones educativas, entre las que se destaca la Universidad Tecnológica Nacional, la Universidad Nacional de Quilmes y la Universidad Nacional de San Martín.

Algunas de estas experiencias y recursos, entre otros, son recomendados por *Program.ar*, un ente del Estado Nacional de Argentina que procura acercar a los jóvenes a la programación y las ciencias de la computación en general, a la que vez que trabaja en la concientización social sobre su importancia.

Entre otros materiales y sugerencias, presenta una serie de criterios que desmitifican ciertas visiones construidas social y educativamente sobre la enseñanza de la computación, que guardan coherencia con el enfoque del presente trabajo. Pese a que están orientados a la educación media y básica, amerita también una lectura desde la realidad universitaria:

- Frente a cierta tendencia respecto que enseñar computación consiste en enseñar a usar paquetes de software para diferentes tareas cotidianas, educativas, creativas, comunicativas, etc. se propone la enseñanza de "algoritmos, modelos, formas de representación de la información, lenguajes, programación, etc". [7]
- Hay veces que se aborda el tema con prejuicios acerca de que las Ciencias de la Computación, en tanto algoritmos, modelos, representaciones y abstracciones, constituyen un contenido muy difícil de aprender. En cambio, se sostiene que "la enseñanza de los conceptos y competencias centrales de computación se puede hacer en forma espiralada"[8], como por ejemplo el algoritmo de la suma, en matemáticas, que se aprende desde temprana edad.
- Con una actitud entre conservadora y desconocedora se encasilla bajo una misma etiqueta de "tecnología" una gran variedad de fenómenos y no es raro escuchar que se necesita volver a cuestiones fundamentales como la lectura, la escritura y las matemáticas, diagnosticando que aprender las tecnologías modernas empeora el rendimiento de dichas habilidades básicas. En contrapartida, desde un conocimiento más preciso de lo que implica aprender Ciencias de la Computación, se afirma que "desarrolla el pensamiento computacional, lo que también puede mejorar el rendimiento en otras áreas, especialmente en el pensamiento matemático que es en donde tiene su origen". [9]

## 3. Mumuki

Frente a la dificultad que se le presenta al estudiante dar los primeros pasos con un lenguaje de programación y su un entorno de desarrollo de software, *Mumuki* es una plataforma web interactiva, que propone un camino progresivo de aprendizaje en conceptos de programación, mediante diferentes lenguajes.

A continuación, se analiza la herramienta desde las categorías presentadas inicialmente.

### 3.1. Énfasis en los conceptos

*Mumuki* propone aprender conceptos de programación en un recorrido conformado por guías de ejercicios donde se combina la explicación de conceptos teóricos con su aplicación práctica. Se presenta al estudiante como una aplicación web en la que el estudiante puede realizar una solución a un problema planteado y ésta es probada y corregida automáticamente, informando acerca de los aciertos y errores.

Las guías de ejercicios mantienen un hilo conductor en el que mediante su resolución progresiva se van incorporando los conceptos. Cada ejercicio tiene al final su conclusión y da pie al siguiente. Se destacan dos estilos de guías, que apuntan a momentos diferentes del proceso de aprendizaje del estudiante.

Las guías de *aprendizaje* son un material interactivo donde se focaliza en presentar conceptos nuevos mediante la resolución de problemas simples, destacándose los corolarios a los que se llega luego de haberlos completado y por un cuidado especial por las validaciones conceptuales que se realizan en la corrección automática.

Las *guías de desafío son* colecciones de problemas de complejidad creciente orientados a afianzar los conceptos aprendidos y ponerlos en práctica en diferentes contextos. Suelen plantear un dominio de problema más amplio, con desafíos que también apelan al ingenio y la creatividad personal.

Si bien hay una secuencia diseñada previamente, en definitiva, es el estudiante quien decide la forma de uso, ya que además no es un requisito haber realizado correctamente un ejercicio para resolver el siguiente. Esta flexibilidad permite también que cada docente plantee su propio recorrido conceptual.

Al elegir un ejercicio, el alumno se encuentra con una explicación donde se enuncia el problema a resolver y se habilita un panel para escribir allí el código de la solución y si el estudiante lo activa, se despliega una ayuda adicional. Al termina de plantear su solución, se debe presionar un botón y la plataforma la ejecuta y evalúa.


**Fig 1: Un ejercicio correctamente resuelto**

En caso que tenga errores, se indica cuáles son en una forma entendible, más sencilla que la que típicamente arroja un compilador -que está orientado a programadores experimentados- y con otros elementos en función de cada ejercicio. Esos errores pueden referirse a tres tipos de problemas:

- Los más básicos son los que hacen que el programa escrito directamente no funcione, es decir, que cuando se interactúa con el compilador del lenguaje correspondiente, éste advierte de los errores y *Mumuki* los adapta para mostrarlos.
- Otros reflejan la situación en la que el programa funcional, pero los resultados no son los esperados. Para ello, internamente se corren una serie de *tests* que contemplan diferentes casos de uso más, con lote de datos representativos de las diferentes aristas del problema.
- Por último, estás los errores conceptuales, donde a pesar que el código funcione y los resultados sean los esperados, la solución no aplica correctamente los conceptos que se busca enseñar mediante el ejercicio.

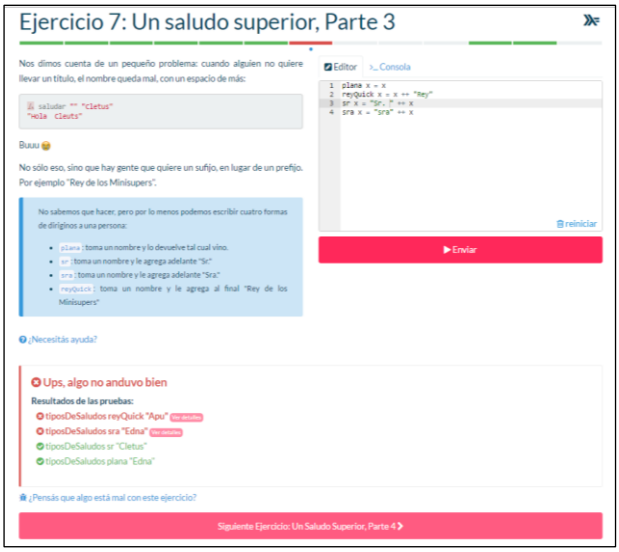


**Fig 2: Un ejercicio que presenta errores**

En todos los casos, se da la posibilidad que el alumno corrija su solución y la vuelva a enviar tantas veces como sea necesario hasta que esté correcta.

## 3.2. El proceso de desarrollo de software

A diferencia de otras herramientas cuyo mismo entorno ya orienta a adquirir habilidades propias del desarrollo de software, el modo de presentación y el entorno de trabajo que propone *Mumuki* es intencionalmente más sencillo y genérico. Abstrayéndose por un momento del contenido, la plataforma no apunta directamente a construir software ni a programar, sino a probar automáticamente soluciones a problemas. Si se construyera por ejemplo un componente apropiado de software que analizara problemas de ajedrez o matemáticas, la interfaz que vería el estudiante, las opciones disponibles y su usabilidad sería prácticamente la misma.

De esta manera, la tensión entre programar y construir software se resuelve a partir del contenido que se genere, en la medida en que incorpora temáticas específicas, como *testing*, *excepciones*, o ejercicios de modelado. Parte del material actualmente disponible apunta a un estudiante inicial, donde generalmente no está presente la inquietud de saber programación como forma de construcción de software. Otra parte -mayoritaria- se orienta a un usuario con perfil de estudiante universitario de carreras de sistemas, que aún siendo de las materias iniciales ya está encaminado en un uso profesional de los lenguajes de programación.

Entendiendo el proceso de construcción de software como más amplio que la tarea de programar, hay puntualmente algunos aspectos propios del proceso de desarrollo, como el análisis del problema, el diseño de la solución, y las pruebas, que si bien en ciertos modelos esquemáticos clásicos se presentan diferenciados de la programación en sí, en numerosas guías de ejercicios *Mumuki* se encuentran integrados.

### 3.3. Priorizando los primeros pasos

Ciertamente, *Mumuki* está orientado a quien está dando los primeros pasos en programación, y su versatilidad en cuanto a la generación de contenido permite que sea utilizado en diferentes contextos educativos. Actualmente, su uso más intensivo y con mayor cantidad de contenido generado es en el marco de trayectos curriculares que apuntan a la formación de un profesional de sistemas, pero también se lo utiliza con el fin de introducir una noción general acerca de la programación, favoreciendo el pensamiento lógico o la capacidad de abstracción, como sucede por ejemplo en los colegios secundarios donde se lo utiliza.

Con este enfoque, una clave de *Mumuki* es la generación del contenido, cómo se plantea la gradualidad en la incorporación de complejidad, lo que la plataforma permite y puede facilitar con sus opciones estructurales, pero que en definitiva es responsabilidad de quien desarrolle el contenido. Actualmente, la amplia mayoría del contenido fue generada por los mismos integrantes del proyecto y un equipo de docentes cercano, con fluidos vínculos laborales e institucionales. En la medida que el proyecto vaya creciendo en aceptación, es de suponer que el contenido se diversifique y sea más difícil velar por su calidad o estilo.

Otro aspecto importante para llevar adelante un recorrido pedagógico es el seguimiento que el docente va haciendo de los estudiantes para lo cual es fundamental todo tipo de feedback que se obtenga de ellos. Para ello, hay una herramienta de uso exclusivo docente, llamada *Mumuki Classroom*, con la que se pude ver el avance de cada alumno en la resolución de ejercicios propuestos. Permite visualizar de manera organizada la información sobre el progreso de sus estudiantes, llegando al detalle de no solo ver el código correcto de la última solución enviada, sino también todos los intentos intermedios que fue probando

El hecho de ser *on line* y que pueda ser utilizado desde cualquier dispositivo sin necesidad de ningún tipo de instalación le da una accesibilidad más amplia. Complementariamente, se está trabajando en el desarrollo de una versión que funcione en forma local, sin necesidad de conectividad.

### 3.4. Desafíos interactivos

En Mumuki se percibe un esfuerzo por cuidar un aspecto visual claro, proponer una sencilla usabilidad y mantener una interacción ágil. En las sucesivas actualizaciones se ven cada vez más ordenadas las opciones disponibles, con mayor cantidad de alternativas de navegación y con un tiempo de respuesta razonable. Es de esperar que se siga trabajando en esa dirección para facilitar su utilización.

Es interesante destacar que todo el contenido de *Mumuki* está en español y no se trata de traducciones sino contenido original, con una redacción informal orientada a la juventud, con modismos argentinos, preguntas, *emoticones* e imágenes, buscando establecer una cercanía entre el idioma de los problemas y el idioma del estudiante. Asimismo, se traducen al español informal los errores más frecuentes que las herramientas empleadas normalmente presentarían en un inglés técnico.

Especialmente en las guías de desafío se introduce una serie de elementos de carácter lúdico, que lejos de infantilizarlo apuntan a motivar al estudiante a querer realizar los ejercicios y avanzar hacia ejercicios más complejos. Por ejemplo, se plantean ejercicios en clave de *puzzle* que invitan a resolverse o se muestran mensajes de felicitación a medida que se van resolviendo los ejercicios. Un aspecto en proyecto por el equipo de desarrollo es precisamente dar más pasos en la *gamificación* de la herramienta.

Una de los lenguajes donde se aprecia mejor el carácter visual de la herramienta es en el soporte al lenguaje *Gobstones*, que incluye un tablero con círculos de colores que se utiliza como forma de presentar los resultados.

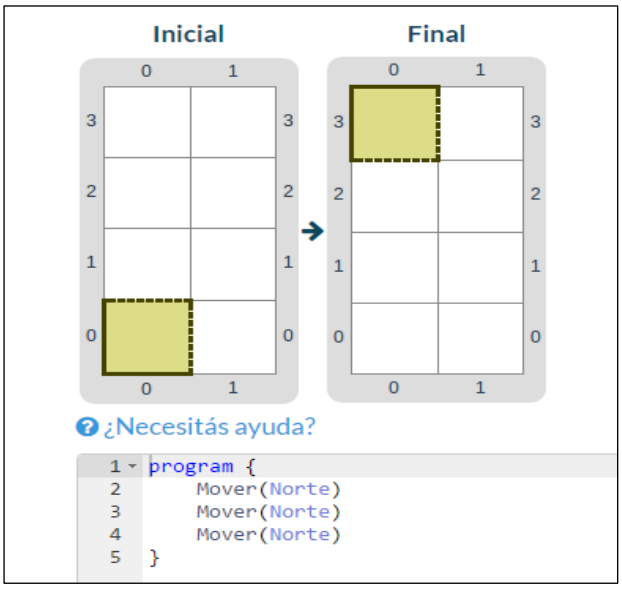


**Fig 3: Interfaz gráfica de *Gobstones* en *Mumuki***

### 3.5. Generación de contenido apropiado

Desde el punto de vista del estudiante la plataforma se muestra como un todo, pero en su configuración se combinan diferentes elementos, organizados en dos grupos:

- ***La plataforma en sí***. Son diversas unidades de software que brindan las prestaciones concretas que provee en cuanto a la lógica de funcionamiento, su interfaz gráfica, el resguardo de las soluciones enviadas, el mecanismo de evaluación, la interacción con los compiladores o intérpretes de cada lenguaje, entre otras. Es la parte eminentemente técnica del proyecto y es software libre y gratuito.
- ***El contenido.*** Es el conjunto de ejercicios disponibles, organizados en guías y libros. Incluye las explicaciones visibles por los estudiantes como así también los *tests* y expectativas conceptuales correspondientes a cada ejercicio. Es la parte más pedagógica del proyecto.

Para la generación de contenido se cuenta con una herramienta específica llamada *Mumuki Editor,* que le permite al docente redactar los enunciados de los ejercicios, plantear las ayudas que le van a aparecer al estudiante cuando la solicite, desarrollar los *tests* y explicitar las expectativas de aplicación de conceptos en la resolución del problema. Esta característica es fundamental para que la plataforma se pueda usar en diferentes instituciones educativos, independizando la tarea de creación de contenido de lo que significa el mantenimiento de la plataforma como tal.

En este esquema, la forma esperada de apropiación de la herramienta por parte de los docentes que la utilizan pasa por la generación o adaptación de contenido acorde a la forma en que cada docente plantea el desarrollo curricular de su asignatura. En el doble rol de docentes y desarrolladores, los responsables del proyecto proveen un camino conceptual mediante guías predefinidas para cada paradigma de programación, pero a la vez, la plataforma permite configurar recorridos específicos para cada universidad, materia u organización que quiera hacer uso de ella.

Actualmente existen ocho universidades e instituciones que lo utilizan, la mayoría de ellas con una configuración específica del contenido. Hay contenido para enseñar programación lógica mediante el lenguaje *Prolog*, programación funcional con *Haskell* y programación orientada a objetos con *Wollok* y *Ruby*. A su vez, como introducción a la programación, se encuentra *Gobstones*.

## 4. Wollok

Retomando los criterios planteados inicialmente como categorías de análisis sobre *Wollok* como herramienta educativa, se puede observar lo que se presenta a continuación.

### 4.1. Conceptos centrales de Programación Orientada a Objetos

Desde la pedagogía que sustenta la herramienta, el acento está puesto en la comprensión de los conceptos de programación orientada a objetos, en la que el lenguaje no es el fin, sino un medio.

El primer concepto sobre el que *Wollok* permite al docente hacer hincapié es precisamente en el de objeto por sobre el de clase, que se basa en la intención de destacar las ventajas del paradigma de objetos evitando ciertas inercias imperativas que llevan a aferrarse al concepto de clase como una simple estructura de datos. Para ello, permite programar fluidamente con objetos sin necesidad de recurrir al concepto de clase y está pensado para un ordenamiento de la propuesta temática que comienza enseñando objetos y trabajando intensivamente con ellos antes que pasar al concepto de clase, dentro de un planteo de incorporación progresiva de complejidad.

Por otra parte, un objeto definido como *objeto bien conocido* no tiene que necesariamente reconvertirse como instancia de una clase, sino que en un sistema complejo puede convivir con el resto de la solución. Es una posibilidad que muchos lenguajes no tienen, sino que fuerzan a que todo el comportamiento se generalice mediante el uso de clases, pero también hay otros lenguajes de objetos que trabajan sólo con objetos, o que combinan clases y objetos, o que utilizan otros conceptos como el de prototipo. En este sentido, *Wollok* introduce a conceptos que se encuentran en diferentes herramientas profesionales de desarrollo dentro del amplio panorama actual. El manejo que ofrece de *mixins* como otra forma de compartir y organizar código entre diferentes objetos va en la misma dirección.

Una facilidad que ofrece el lenguaje, útil en los primeros ejemplos en los que se pretende aprender a modelar objetos y mensajes, es la posibilidad que los objetos tengan valores iniciales para sus variables sin que se les envíe ningún mensaje previamente.

Uno de los conceptos distintivos del paradigma, que ya estaba presente en los primeros lenguajes de objetos y que *Wollok* mantiene, es el de *polimorfismo*, que gracias al mecanismo de *tipado dinámico* permite un manejo sencillo y transparente, sin necesidad de declaraciones ni de necesitar *herencia* o *clases*. La convicción acerca de la importancia del *polimorfismo* para construir soluciones informáticas profesionales lleva a que se permita explicar

el concepto en las primeras clases y al avanzar con la cursada se lo siga aplicando en situaciones más complejas y se lo articule con los siguientes conceptos.

## 4.2. Una dinámica de construcción de software

Retomando la idea de pensar la herramienta como una totalidad junto con su entorno de desarrollo más que como un lenguaje, *Wollok* se puede definir como un *SDK* (Software Development Kit) que percibe un acercamiento a la dinámica de trabajo que se usa en la actualidad.

A diferencia de *Smalltak*, el lenguaje originario del paradigma de objetos, que consta de un entorno propio muy diferente a la mayoría de los lenguajes y que prácticamente es el único que trabaja con una imagen *viva*, *Wollok* trabaja sobre archivos, viene integrado en el entorno Eclipse -muy utilizado en la actualidad- y tiene un estilo similar a los lenguajes más difundidos.

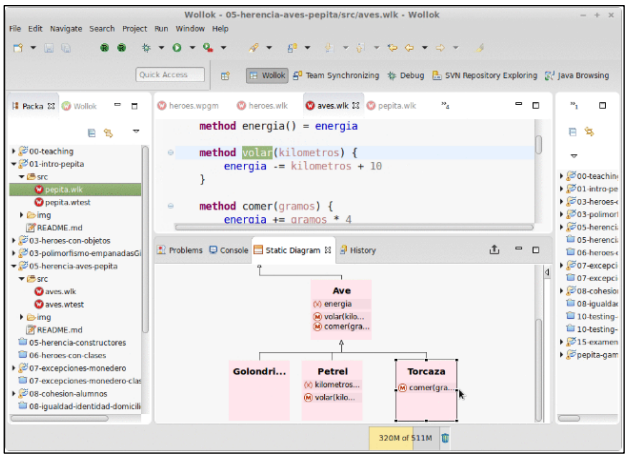


**Fig 4: Ambiente de desarrollo de Wollok**

El entorno de desarrollo está configurado de tal manera que se aprovechan características estándar que agilizan la tarea de desarrollo, como modalidades de edición y navegación en el código, se ocultan ciertas opciones para evitar complejidad innecesaria, a la vez que se incorporan herramientas adicionales que ayudan al proceso de construcción de software. Por ejemplo, cuenta con una herramienta para organizar y evaluar *tests unitarios*, asociada a una determinada sintaxis para construirlos, consta de una consola para pruebas puntuales y un *debugger* (depurador de errores) y consola de pruebas. También facilidades para la edición del código, como coloreado de sintaxis, opciones de autocompletado y *qick-fixes* (ajustes rápidos). Son todas herramientas que le permiten al estudiante ir introduciéndose en un estilo de trabajo profesional, donde más allá que el lenguaje *Wollok* como tal difícilmente sea utilizado laboralmente, se aprovechan no sólo los conceptos de programación, sino también un conjunto de habilidades y destrezas específicas que no forman parte del curriculum prescripto, pero que apuntan a educar en competencias.

## 4.3. Hacia una práctica profesional

El contexto profesional actual presenta un abanico amplio de lenguajes de programación orientada a objetos, algunos con mayor nivel de utilización y otros no tanto, algunos más clásicos y otros con conceptos innovadores, algunos con un planteo más "puro" respecto del paradigma y otros con más "híbridos", con influencias de otros paradigmas de programación.

Siendo una herramienta educativa y además de reciente creación, es claro que *Wollok* como tal no tiene un uso directo en el ambiente profesional, pero entendiendo que la inserción laboral es el punto de llegada de la propuesta educativa y no de partida, la clave es ver en qué medida esta herramienta permite u favorece que el estudiante vaya progresivamente adquiriendo habilidades, capacidades y conceptos propios de la profesión.

*Wollok*, antes que pretender conservar las ideas originarias que remite al momento fundante del paradigma, busca ser representativo de lo que en la actualidad se entiende por la programación orientada a objetos. A la vez que evita incluir ideas procedurales o de otros paradigmas que alteran el modelo de objetos, incluye conceptos novedosos que van más allá del modelo de objetos clásico y que forman parte del complejo y diverso presente del paradigma. En particular, amplía precisamente el sentido del concepto de *clase* como única forma de definir comportamiento para los objetos, ya sea con *objetos bien conocidos* como con *mixins*.

## 4.4. Diagramas y juegos

Más allá que el mismo entorno de desarrollo *Wollok* tiene características que lo hacen relativamente sencillo de manejar y con una usabilidad aceptable, dentro de los estándares utilizados en la industria de desarrollo, hay algunas herramientas en particular que se destacan en esta tensión por aprovechar los recursos gráficos y lúdicos con criterio didáctico sin perder profesionalismo.

Una herramienta que se destaca en este sentido es la que genera diagramas que se generan automáticamente a partir del código y permiten ver más claramente la presencia de ciertos conceptos, como por ejemplo el manejo de referencias, y ayudan a visibilizar el diseño de una solución. Concretamente, se cuenta con un diagrama de *clases* y otro de *objetos*. Cabe reconocer que desde el punto de vista estético se podría mejorar y que podría tener mayor grado de interactividad en sintonía con la ejecución

de las pruebas, a la vez que está presente entre los trabajos pendientes para futuras actualizaciones.

Otra herramienta que viene integrada es lo que se denomina *Wollok Game*, que consiste en una biblioteca con funcionalidades disponibles para generar juegos y en general aplicaciones gráficas interactivas. Desde el punto de vista de la implementación se trata de un conjunto de *objetos* que entienden *polimórficamente* determinados *mensajes*, por lo que el estudiante los puede usar como parte de su proceso de aprendizaje de los conceptos del paradigma. Entrando en su aspecto visual, se trata básicamente de un tablero en el que se desplazan diferentes personajes a partir de la interacción con el usuario. A su vez, se cuenta con algunos ejemplos de juegos ya implementados.

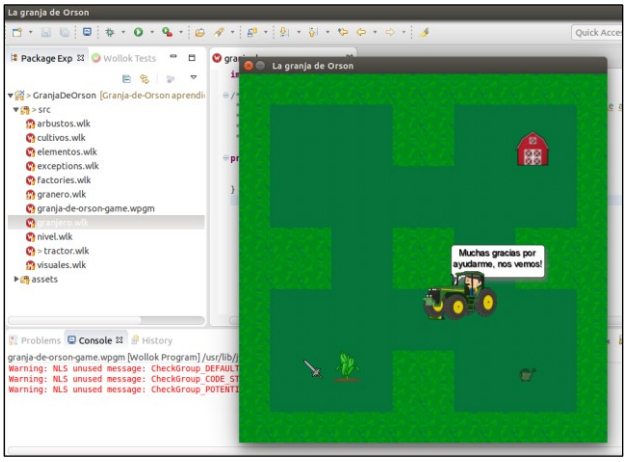


**Fig 5: Ejemplo de Wollok Game**

### 4.5. Desarrollo y utilización de herramientas propias

En primer lugar, *Wollok* es software libre. Su código de está completamente disponible, tanto el *core* del sistema que está desarrollado en *java/xtext*, como sus bibliotecas que están implementada en el mismo *Wollok*. La tendencia que se percibe, a partir de los cambios internos en las sucesivas versiones que fueron poniéndose a disposición, apunta a disminuir la proporción de métodos nativos en favor de aumentar el código *Wollok*.

El equipo de desarrollo del software mantiene actualizada la herramienta, que si bien ya está siendo utilizada en diferentes universidades va ampliando sus prestaciones y adaptándose a los pedidos y observaciones de los docentes que lo utilizan. Ciertamente, el hecho de que los desarrolladores también sean docentes y que compartan vínculos y espacios institucionales con el resto de los educadores que también utiliza la herramienta, facilita este intercambio que al equipo de desarrollo le sirve de retroalimentación y a los docentes les permite tener una herramienta que se incorpora mejor dentro de su propuesta pedagógica. Actualmente, teniendo en cuenta que *Wollok* es utilizado en diferentes materias de carreras de sistemas de tres universidades, lo que se traduce en alrededor de veinte comisiones con sus respectivos docentes, el intercambio se sostiene fluidamente y se puede afirmar que hay una fuerte apropiación de la herramienta por parte de sus usuarios.

En otras palabras, el hecho de ser un software educativo creado por docentes permite un dinamismo particular a la hora de definir los alcances y características del lenguaje, acorde a una determinada forma de percibir el contexto profesional y de asumir la tarea educativa.

## 5. Conclusiones

Ciertamente, los criterios planteados no son los únicos posibles y lejos se está de pretender invalidar otras categorías de análisis. Lo que sí se constata es que resultan útiles para abordar las dos experiencias presentadas y que serían factibles de extender a las otras experiencias también enumeradas.

Como se mencionó oportunamente, de ninguna manera se podría afirmar que de por sí garanticen los resultados esperados, sino que, en tanto herramientas, dependen de la forma de utilización propuesta por parte de los docentes y de su inclusión en un recorrido curricular en el que adquieren sentido.

En ambos casos, aún con las limitaciones propias de ser software educativo de reciente creación, que continúa redefiniéndose y cuya utilización está brindando las primeras experiencias situadas, a partir de lo analizado en tanto herramientas se constata que presenta características que habilitan a ser utilizadas eficazmente como recurso pedagógico para aprender a programar.

Lo que se observa es que, por la forma en que están construidas, por la interfaz que presentan al estudiante, por la posibilidad que ofrecen para la focalización en los conceptos, por no presentar obstáculos para un estudiante sin experiencia previa, por la selección de elementos teóricos que brinda acorde al contexto, por las competencias a las que alude, se trata de herramientas que a la vez que resultan propicias para dar los primeros pasos en la programación permiten ser utilizadas para la formación de profesionales en sistemas que se desarrollen laboralmente en el ambiente actual de desarrollo de software.

Como línea de investigación pendiente para continuar la temática, se propone hacer trabajos de campo que den cuenta de la utilización de estos recursos por parte de grupos de estudiantes en el marco de propuestas pedagógicas que las contextualicen. Desde esta perspectiva, se podrían establecer conclusiones recuperando la perspectiva de los estudiantes y docentes

participantes en tanto protagonistas del proceso de enseñanza aprendizaje.

Por otra parte, cabe señalar nuevamente lo singular de la confluencia en tanto docentes y desarrolladores de la herramienta, lo que facilita su mutua adecuación y retroalimentación. En este sentido, no se busca una generalización o extrapolación mecánica de estas conclusiones, sino que binde elementos para analizar la significatividad de la utilización pedagógica de estos recursos en sus respectivos contextos.

## 6. Referencias


[1] Freire, Paulo. (1994) *Carta a quien pretende enseñar.* Ed. Siglo Veintiuno. Buenos Aires.
[2] Ibíd.
[3] Vigotsky, Lev. (1993) *Pensamiento y lenguaje.* En Obras Escogidas. T 2 Ed. Visor. Madrid
[4] Bruner, Jerome. (1988) *Realidad mental y mundos posibles.* Ed. Gedisa. Barcelona.
[5] Bixio, Cecilia (2010) *Maestros del siglo XXI. El oficio de educar. Homenaje a Paulo Freire.* Ediciones Homosapiens. Rosario.
[6] Resnick, Mitchel. "Reviving Papert's Dream" en *Educational Technology.* Vol 52. N° 4. Julio-Agosto 2012.
[7] Program.ar (2004) *Por qué todos tienen que aprender a programar.* Ministerio de Ciencia Tecnología e Innovación Productiva, Ministerio de Educación, Fundación Sadosky.
[8] Ibíd
[9] Ibíd