UNIVERSIDAD
COMPLUTENSE
MADRID

Proyecto de Innovación y Mejora de la Calidad Docente

Convocatoria 2014

Proyecto 97/2014

Estudio de viabilidad de un entorno de aprendizaje colaborativo de lenguajes de programación

Responsable: Adrián Riesco Rodríguez

Facultad de Informática

Departamento de Sistemas Informáticos y Computación

1. Objetivos propuestos en la presentación del proyecto

En este proyecto pretendemos estudiar la viabilidad de un proyecto colaborativo para enseñar lenguajes de programación. La idea principal, similar a la desarrollada en Duolingo (`https://www.duolingo.com/`), es compartir los conocimientos de usuarios expertos en cierto lenguaje para enseñar a otros usuarios que, a su vez, son expertos en el lenguaje que los primeros usuarios quieren aprender. De esta manera potenciamos el aprendizaje, por un lado, en modo "flipped classroom", en el cual son los propios compañeros, con los que los alumnos sienten más cercanía, quienes explican los conceptos básicos del lenguaje, dejando al profesor la tarea de comprobar la corrección de la información proporcionada por cada uno. Permite además nivelar el conocimiento de los estudiantes, pues comparten ciertos conocimientos mientras obtienen otros.

Este aprendizaje se realizaría a través de ejemplos cortos que puedan ser traducidos desde el lenguaje en el que el usuario es experto al lenguaje que el usuario quiere aprender. Asimismo, deberá estudiar las traducciones realizadas en sentido contrario por otros usuarios. Esta aproximación busca un sistema modular de formación, dado que es necesario delimitar claramente las tareas para que sea más sencillo llevarlas a cabo de manera correcta. Esta división en módulos también facilita la adquisición y evaluación de las competencias genéricas y transversales, pues se pueden localizar fácilmente los errores concretos cometidos por cada alumno.

Si, como creemos, consideramos que el proyecto es viable y útil para un número razonable de asignaturas (centrados en las facultades de Informática, Ciencias Matemáticas y Estudios Estadísticos), entonces estudiaremos las distintas fases en las que debe llevarse a cabo y los recursos necesarios para ello. En concreto, estamos interesados en saber cuáles son los módulos generales en los que se puede dividir el aprendizaje de un lenguaje de programación, generar ejemplos típicos en la enseñanza de cualquier lenguaje de programación y añadir lenguajes partiendo desde cero.

Además, para permitir la inmersión de los estudiantes en el sistema pretendemos estudiar su desarrollo para plataformas móviles, su integración con redes sociales, incluyendo una serie de hitos que les permita mostrar sus resultados (siguiendo la idea de las "insignias"), y su integración con el Campus Virtual.

En definitiva, pretendemos sentar las bases para desarrollar un entorno ágil, en el que los estudiantes puedan centrarse en tareas concretas a la vez que ayudan a sus compañeros. Este sistema, cuyo último estadio de desarrollo proporcionaría una plataforma con material altamente extensible y reutilizable, potenciaría la participación activa del alumno en la enseñanza ya que, por un lado, se encuentra en un entorno de evaluación continua en el que es "juzgado" por pares, lo que no supone una interacción, siempre más complicada y "definitiva", con el profesor. Por otro lado, participa activamente en los contenidos de la asignatura, lo que le hace seguir la misma con más interés.

2. Objetivos alcanzados

Consideramos que el proyecto es viable y que puede reportar diversos beneficios tanto a estudiantes como a docentes. En esta sección justificamos su adecuación a la docencia, mientras que en la sección 6 presentamos un posible diseño, futuras extensiones y diferentes usos.

2.1. Aspectos beneficiosos que aporta a los estudiantes

Todos los docentes con experiencia en la enseñanza de lenguajes de programación están familiarizados con las dificultades a las que se enfrentan los alumnos en su aprendizaje, incluso si se trata de alumnos con un alto grado de motivación, como suele ser el caso de los alumnos en este tipo de asignaturas. La motivación inicial surgida ante la perspectiva de estar construyendo artefactos que "funcionan", como es el caso de un programa correcto, se ve en muchas ocasiones confrontada a la frustración por la inherente dificultad de la tarea de programar, debido en gran parte al carácter discreto o discontinuo en dicha corrección. Es decir, pequeños cambios en el programa pueden suponer errores catastróficos. Esta naturaleza discreta en la resolución de problemas es para los alumnos algo completamente nuevo y en parte el motivo de la frustración antes mencionada.

Otra dificultad a la que se enfrentan los alumnos es la de identificar los errores que tienen que ver con la corrección no funcional, es decir, errores presentes en programas que, a pesar de ellos, "funcionan" como deberían. Resulta complicado para el docente llevar un seguimiento exhaustivo de los errores de este tipo. Creemos que una herramienta colaborativa al estilo Duolingo puede aportar mejoras en el aprendizaje de los alumnos, y en particular respecto de las dificultades comentadas antes. La interacción con otros alumnos en particular pretende fomentar la discusión acerca de características no funcionales de los programas. Además, pensamos que el aprendizaje mediante ejemplos preconizado por Duolingo puede complementar la enseñanza tradicional no solo de lenguajes de programación, sino también de métodos de resolución de problemas algorítmicos. Por último, creemos que esta herramienta puede ayudar a identificar y corregir de manera temprana problemas que, en ocasiones, se detectan demasiado tarde y lastran el aprendizaje.

2.2. Asignaturas en las cuales puede ser beneficioso el proyecto

En este proyecto hemos analizado las asignaturas impartidas por nuestro departamento en distintas titulaciones para identificar aquellas en las que estimamos que nuestra herramienta podría resultar beneficiosa para el aprendizaje de los alumnos. A grandes rasgos, podemos hablar de dos categorías de asignaturas:

(A) asignaturas en las que dentro de los objetivos formativos se contempla el aprendizaje de un nuevo lenguaje de programación;

(B) asignaturas en las que se estudian métodos y técnicas para la resolución de problemas usando algún (o cualquier) lenguaje de programación previamente conocido por los alumnos.

En las tablas 1 y 2 pueden verse las asignaturas en las categorías A y B, respectivamente, que hemos identificado que pueden ser beneficiadas por este proyecto. Dentro de la categoría A destacamos las asignaturas de Fundamentos de la Programación y Tecnología de la Programación. Ambas son asignaturas obligatorias de los primeros cursos impartidas en todos los grados de la Facultad de Informática. Durante este proyecto hemos mantenido reuniones con profesores de ambas asignaturas, los cuales han mostrado su disposición a usar una herramienta de este tipo.

La asignatura Fundamentos de Programación se cursa en el primer curso. Se trata de una asignatura introductoria a los conceptos básicos de la programación estructurada en un lenguaje imperativo, usando para ello el lenguaje C++. En todas las titulaciones es de las asignaturas que cuenta con un menor índice de aprobados con respecto al número de alumnos matriculados. En este caso, necesitaremos implementar la característica de traducción desde pseudocódigo, ya que los alumnos en su gran

ASIGNATURA	CURSO	TITULACIÓN
Fundamentos de la Programación	1	Grado en Ing. Informática, Grado en Ing. del Software, Grado en Ing. de Computadores
Informática	1	Grado en Matemáticas, Grado en Ing. Matemáticas, Grado en Matemáticas y Estadística
Tecnología de la Programación	2	Grado en Ing. Informática, Grado en Ing. del Software, Grado en Ing. de Computadores
Programación Declarativa	3	Grado en Ing. Informática
Programación Declarativa Aplicada	2	Máster en Ing. Informática
Programación Declarativa	4	Grado en Matemáticas

Cuadro 1: Asignaturas en la categoría A

ASIGNATURA	CURSO	TITULACIÓN
Estructuras de Datos y Algoritmos	2	Grado en Ing. Informática, Grado en Ing. del Software, Grado en Ing. de Computadores
Métodos Algorítmicos en Resolución de Problemas	3	Grado en Ing. Informática
Diseño de Algoritmos	3	Grado en Ing. de Computadores
Técnicas Algorítmicas en Ingeniería del Software	3	Grado en Ing. del Software
Estructuras de Datos	4	Grado en Matemáticas
Diseño de Algoritmos	4	Grado en Matemáticas

Cuadro 2: Asignaturas en la categoría B

mayoría no cuentan con conocimientos previos de ningún lenguaje de programación (haremos hincapié en esta funcionalidad de nuestra herramienta en la sección 6.7). Creemos que nuestra herramienta puede ayudar en buena medida en el proceso de aprendizaje de los alumnos en esta asignatura. Por un lado, al tratarse de una asignatura de primer curso impartida en todos los grados, el número de alumnos matriculados es muy elevado (alrededor de 570 alumnos en el curso 2014/15), lo que la hace especialmente adecuada para el aprendizaje colaborativo. Por otro lado, y también por el mismo motivo, los tamaños de los grupos son significativamente mayores que en otros cursos, lo que, unido al hecho de que se trata de alumnos de nuevo ingreso, hace que en ocasiones el trato con el profesor no sea suficientemente cercano. Esperamos que en este contexto una herramienta como la que proponemos en este proyecto estimule el intercambio de conocimiento entre los alumnos.

El caso de la asignatura Tecnología de la Programación difiere de Fundamentos de la Programación al tratarse de una asignatura de 2º curso, aunque también obligatoria en todos los grados impartidos en la facultad. Se trata de una asignatura de programación avanzada, introduciendo conceptos típicos de la Programación Orientada a Objetos. En este caso el lenguaje utilizado es Java, si bien el lenguaje conocido por los alumnos hasta el momento es únicamente C++. Por ello, pensamos que nuestra herramienta puede ayudar a los alumnos en esta asignatura en dos sentidos: (i) puede ayudar a los alumnos a aprender Java desde C++, antes de adentrarse en los nuevos conceptos de programación que se imparten en la asignatura, y (ii) puede estimular a los alumnos interesados a aplicar esos conceptos avanzados aprendidos en Java de vuelta a C++.

En cuanto a la categoría B, destacamos las asignaturas de Estructuras de Datos y Algoritmos, impartida en todos los grados, y Métodos Algorítmicos en Resolución de Problemas, impartida en el Grado en Ingeniería Informática, o sus análogas en el Grado en Ingeniería del Software y en el Grado en Ingeniería de Computadores, Técnicas Algorítmicas en Ingeniería del Software y Diseño de Algoritmos, respectivamente. En estas asignaturas se pondrían en práctica las dos últimas lecciones en las que se divide el aprendizaje de un lenguaje de programación (ver sección 6.3).

3. Metodología empleada en el proyecto

Podemos distinguir 5 tareas en este proyecto:

1. Revisión del estado del arte.
2. Aplicación del sistema a la docencia.
3. Integración con otros sistemas.
4. Captura de requisitos para la implementación.
5. Estudio de otros beneficios del sistema.

El estudio de viabilidad comenzó con una revisión del estado del arte, cuyos resultados se presentan en la sección 6.1. En primer lugar se estudiaron los jueces automáticos, pues como se indica en la sección 4 contábamos con un experto en la materia en el equipo. También se revisaron herramientas similares para el aprendizaje colaborativo como Duolingo, que sirve para aprender idiomas a través de traducciones. Esta revisión del estado del arte incluye las publicaciones científicas más relevantes, tanto en el campo del aprendizaje colaborativo como del aprendizaje de los lenguajes de programación.

En la Tarea 2 estudiamos la aplicación del sistema propuesto a la docencia. Para ello revisamos en detalle los planes de estudio y las fichas docentes de los grados y posgrados en los que tiene presencia nuestro departamento (Fac. Informática, Ciencias Matemáticas y Estudios Estadísticos). Una vez localizadas las asignaturas en las que se enseñan lenguajes de programación nuevos, concertamos reuniones con los profesores (principalmente los coordinadores) para conocer el enfoque seguido para el aprendizaje y valorar con ellos lo que una herramienta de aprendizaje colaborativo podría aportar. Se pueden ver los resultados de esta tarea en la sección 2.

En la Tarea 3 consideramos las posibilidades de integración con otras herramientas. Se dio prioridad al Campus Virtual, ya que es la que más impacto tendría en la docencia de la UCM. Para ello se estudiaron las APIs de integración de Sakai y Moodle, comprobando las dificultades técnicas. Se estudió también la posibilidad de desarrollar apps para móviles, y las APIs de las principales redes sociales (Facebook, Twitter o Google+) para gestionar log-ins y publicar resultados. Los resultados de esta fase se encuentran en la sección 6.6.

En la Tarea 4 capturamos los requisitos (software, hardware y personal) necesarios para desarrollar una herramienta de aprendizaje colaborativo. Se aplicaron técnicas de Ingeniería del Software para aproximar el esfuerzo necesario y obtener un plan de proyecto. En este punto buscamos la opinión de profesores de nuestro entorno con experiencia en el desarrollo de herramientas web de aprendizaje. Para la captura de requisitos utilizamos los comentarios de profesores obtenidos en la Tarea 2, que se completaron en reuniones plenarias. Una vez identificados, se realizó un informe, disponible en la secciones 6.2 y 6.3, con los principales requisitos y un plan de proyecto aproximado. Asimismo, proponemos en las secciones 6.4, 6.5, 6.7 y 6.8 futuras mejoras una vez el núcleo del sistema esté operativo.

La Tarea 5 se centró en otros beneficios del sistema fuera del ámbito educativo. Para ello se revisaron herramientas comerciales similares (p.ej. Duolingo y UVA Online Judge) para conocer su modelo de negocio. En la sección 6.9 se proponen ideas para futuros usos.

4. Recursos humanos

Este proyecto fue desarrollado por los doctores Enrique Martín, Manuel Montenegro, Adrián Riesco, Fernando Rosa y Salvador Tamarit. El trabajo en las distintas tareas se distribuyó como sigue:

Tarea 1: revisión del estado del arte

Responsable: Enrique Martín.

Participantes: Manuel Montenegro, Adrián Riesco, Fernando Rosa y Salvador Tamarit.

Esta tarea fue la primera en realizarse ya que es la base de las siguientes. Nos permitió conocer otras herramientas existentes y enfoques de aprendizaje colaborativo aplicados anteriormente. De ahí obtuvimos experiencias (tanto positivas como negativas) y nuevas ideas. El responsable fue Enrique Martín por su experiencia con herramientas web para el aprendizaje web en anteriores PIMCDs, aunque todos los miembros del equipo participaron, ya que era necesario que todos conociésemos las tecnologías actuales a partir de las cuales trabajar.

Tarea 2: aplicación del sistema a la docencia

Responsable: Fernando Rosa.

Participantes: Enrique Martín, Manuel Montenegro, Adrián Riesco y Salvador Tamarit.

Esta tarea supuso el resultado de mayor peso del proyecto, ya que supuso revisión de planes de estudio, contactar con profesores, reuniones y estudio en común. El responsable es Fernando Rosa por su mayor experiencia docente en asignaturas de programación en la Facultad de Informática, aunque de nuevo fue necesaria la participación de todo el equipo.

Tarea 3: integración con otros sistemas

Responsable: Manuel Montenegro.

Participantes: Enrique Martín, Adrián Riesco y Fernando Rosa.

Manuel Montenegro fue elegido responsable de la tarea porque cuenta con experiencia en sistemas de gestión de contenidos (o CMS, del inglés Content Management System) y en el desarrollo de plugins para CMS y su integración, además de contar con amplio conocimiento de los sistemas que forman el Campus Virtual de la UCM.

Tarea 4: captura de requisitos para la implementación

Responsable: Salvador Tamarit.

Participantes: Manuel Montenegro y Fernando Rosa.

El responsable fue Salvador Tamarit, que ha participado en el desarrollo de diversas herramientas de tamaño medio y grande, tanto en entorno web como de escritorio.

Tarea 5: estudio de otros beneficios del sistema

Responsable: Adrián Riesco.

Participantes: Enrique Martín y Salvador Tamarit.

Adrián Riesco, responsable de la tarea, es el miembro con mayor experiencia en herramientas de aprendizaje colaborativo. Por ello, pudo guiar el estudio sobre aquellos que, sin ser docentes, más beneficio proporcionarían a la herramienta.

5. Desarrollo de las actividades

Para el desarrollo del proyecto definimos un plan de trabajo basado tanto en los objetivos que se deseaban alcanzar como en las tareas identificadas. Para ello, diseñamos un diagrama de Gantt, presentado en la figura 1, en el cual definimos el plan de trabajo que seguimos. No se incluyen en éste reuniones de coordinación generales y de las tareas. Para las primeras establecimos un intervalo de dos semanas entre reuniones y para las segundas el intervalo fue de una semana.

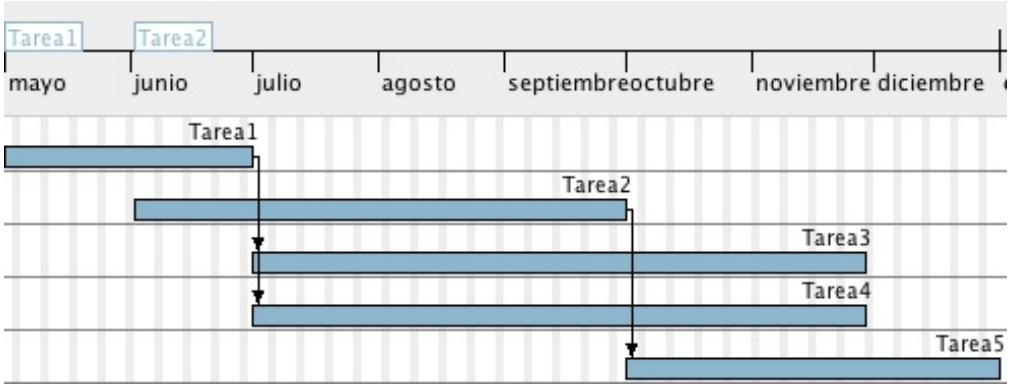

Figura 1: Cronograma del proyecto

Como se puede observar en el diagrama de Gantt, empezamos los primeros dos meses con una revisión del estado del arte que sirvió de base, como se ha explicado en la sección 3, para las siguientes tareas que fueron desarrolladas en los siguientes meses. Centramos las primeras semanas en la revisión de literatura en aspectos de aplicabilidad en la docencia, ya que en junio empezamos a trabajar en la aplicación del sistema a la docencia. Para la consecución de esta primera tarea, se dividieron los recursos a estudiar entre los miembros del equipo y se realizaron reuniones periódicas para poner en común los conocimientos adquiridos, de tal manera que todos obtuvimos una base sólida. También se utilizó un repositorio Subversion para poner en común los distintos recursos encontrados.

Elegimos junio para empezar la segunda tarea por considerar que era la mejor época para realizar este estudio, ya que los docentes tienen, por una parte, la experiencia reciente del curso que acaba de finalizar y, por otra, se empieza a preparar el siguiente curso. De hecho, esta tarea se extenderá hasta finales de septiembre para estudiar su aplicación en el comienzo del nuevo curso. Como hemos indicado en la sección 3, para la Tarea 2 se revisaron en detalle los planes de estudio y se hicieron reuniones con los coordinadores de las asignaturas más relevantes para el proyecto. Los comentarios de los profesores se pusieron a disposición de todos los miembros del proyecto mediante el repositorio común. Finalmente se realizaron una serie de reuniones plenarias en las que se evaluó con detalle el impacto y la aplicación de una herramienta de aprendizaje colaborativo para la programación, tomando en cuenta las opiniones obtenidas de los coordinadores/profesores y la experiencia de los distintos miembros del grupo.

Volviendo al estudio del estado del arte, a los dos meses, una vez finalizada esta tarea y adquirido el conocimiento necesario, se dio partida a dos tareas que debían hacerse en paralelo. Por una parte, estudiar los diferentes sistemas en los que el proyecto se podría integrar, comenzando por investigar la capacidad de integración de cada una de estas. Por otra parte, la segunda tarea, que concurre con la anterior, era la captura de requisitos para la implementación de una herramienta. Es importante destacar que estas tareas debían ejecutarse a la vez, ya que puede que ciertas restricciones deducidas del estudio de integración afectasen a los requisitos de implementación y viceversa. Estas dos tareas fueron las de mayor carga del proyecto, extendiéndose durante cinco meses. Una vez finalizadas ambas, nos centramos en estudiar otros beneficios del sistemas con el conocimiento adquirido en el resto de tareas ya finalizadas. Esta última tarea empezó realmente dos meses antes, con la finalización de la tarea de aplicación del sistema a la docencia, ya que a partir de este punto se tenía una parte del

proyecto finalizada y se tenía por tanto una perspectiva suficiente para empezar a pensar en beneficios complementarios a los observados tanto en las fases previas del proyecto como en el desarrollo de sus tareas.

Este plan de trabajo fue coordinado por el investigador responsable, que se encargó de dar una cobertura global al proyecto, mientras que en cada tarea fue el responsable de ésta el que hizo ese trabajo de coordinación a un nivel más local y concreto.

6. Anexo

En esta sección presentamos una revisión del estado del arte y una guía para facilitar la implementación del sistema en futuros proyectos de innovación.

6.1. Revisión del estado del arte

Los *jueces on-line* o *correctores automáticos* [20, 12, 19] son herramientas que durante bastantes años se han utilizado para el aprendizaje y práctica de lenguajes de programación, tanto en concursos de programación como en la docencia en aula. Estos sistemas se basan en el uso de casos de test ocultos para comprobar la corrección de los ejercicios enviados por los alumnos. Los enunciados de los problemas que contienen los jueces on-line siguen un esquema común:

- Descripción en lenguaje natural de la tarea que se debe resolver.
- Descripción detallada del formato de los datos de entrada. Como estos datos se leerán de ficheros de texto a través de la entrada estándar es muy importante que su formato quede especificado con detalle.
- Descripción de la salida que debe producir el programa del alumno. Al igual que la entrada, es muy importante definir detalladamente este formato ya que cualquier diferencia, por pequeña que sea (espacios, tabuladores, saltos de líneas, mayúsculas/minúsculas) hará que el caso de prueba sea considerado negativo.
- Debido a la importancia de cumplir 100 % con los formatos de entrada y salida, los enunciados también contienen algunos ejemplos de datos de entrada junto con su correspondiente salida.

Cuando un sistema recibe un envío, compila y ejecuta el programa con los distintos casos de prueba. Si la salida obtenida para cada caso es exactamente igual que la esperada[1], el ejercicio se considera superado. Si algún caso de prueba falla, el ejercicio se considera no superado. Los sistemas también producen mensajes si el proceso ha fallado en algún paso, por ejemplo por un error de compilación o un error de ejecución. Debido al peligro de ejecutar código ajeno en el ordenador, los sistemas introducen limitaciones a la hora de ejecutar los programas de los alumnos y por ello producen mensajes cuando estos límites se exceden: tiempo de CPU, memoria utilizada, tamaño de la salida, etc.

Dentro de los jueces on-line orientados a concursos de programación cabe destacar Mooshak [25], el juez on-line de la Universidad de Valladolid [37] y USACO [22]. Todos ellos están preparados para desplegarse en entornos distribuidos de tal manera que la carga de correcciones se pueda balancear, que en algunos momentos de un concurso de programación puede ser bastante alta.

Por otro lado, existen jueces más orientados al apoyo de la docencia en aula como Web-CAT [14], CourseMarker [18], TRAKLA [30] y BOSS [21]. Este grupo es más heterogéneo, ya que poseen características particulares que los diferencian. Por ejemplo, Web-CAT está más enfocado a que los propios alumnos desarrollen sus casos de prueba, y además permite corregir ejercicios inspeccionando el código enviado (por ejemplo para medir la calidad de los comentarios, la estructura, etc.) y no solo la salida generada. CourseMarker proporciona un cuidado sistema de roles para gestionar las distintas tareas la evaluación, además de permitir la corrección de ejercicios basados en diagramas como pueden ser diagramas de flujo, diseños de bases de datos o diseños de circuitos. TRAKLA se centra en la ejecución paso a paso de algoritmos y la visualización de las estructuras asociadas. En lugar de solicitar un código a ejecutar completamente, TRAKLA muestra las estructuras y solicita a los alumnos que apliquen las modificaciones en cada paso del algoritmo. Por último, BOSS permite la corrección de envíos no solo mediante tests sino también a través de la inspección de código para medir su calidad usando distintas métricas y también para la detección de plagios.

[1] Algunos sistemas permiten relajar esta comprobación e ignorar los caracteres blancos como espacios y tabuladores

En un escalón intermedio entre jueces orientados a los concursos cabría destacar *jutge.org* [36], desarrollado en la Universitat Politècnica de Catalunya, y *¡Acepta el reto!* [1], una iniciativa desarrollada en los últimos años en la Universidad Complutense de Madrid. Estos sistemas están disponibles on-line para la práctica de la programación y albergan concursos ocasionalmente, pero también están integrados con distintas asignaturas de dichas universidades, en cuya docencia se utilizan con asiduidad.

En todos los casos anteriores, la evaluación de los envíos de los alumnos en los jueces on-line se basa principalmente en la ejecución de casos de prueba. Ello exige que los instructores desarrollen previamente un conjunto suficiente de pruebas, tarea tediosa y con bastante complejidad ya que estos deben cubrir todos los posibles casos, tantos los usuales como los casos frontera. Además, el uso de casos de prueba requiere de la ejecución de programas de los alumnos en los servidores del corrector, lo que acarrea importantes problemas de seguridad. Para garantizar la integridad del servidor hay que tomar varias medidas de protección: ejecutar el programa en entornos virtuales aislados del resto del sistema de ficheros, limitar el tiempo y la memoria utilizada, restringir los posibles accesos a dispositivos externos como la red, limitar los hilos o procesos que pueden crear... Por último, el uso de casos de prueba limita la precisión de la evaluación ya que solo comprueba la *equivalencia funcional* de los códigos. Por ejemplo, un juez on-line puede contener un ejercicio de ordenación para ejercitar un determinado método (léase *quicksort*). Sin embargo los usuarios podrán enviar programas con métodos diferentes que produzcan ordenaciones correctas y posean una complejidad similar o mejor en tiempo y espacio (léase *mergesort* o *heapsort*) y para el sistema serán indistinguibles: pasan todos los casos de prueba y por tanto son correctos.

Además de los jueces on-line para lenguajes de programación existen otras herramientas que tienen ciertos parecidos a la nuestra. Una de ellas es Doulingo [13], una herramienta de aprendizaje de idiomas y también una herramienta de traducción de textos mediante *crowdsourcing* (i.e. colaboración abierta distribuida o externalización abierta de tareas). Las caraterísticas más interesantes de su modelo educacional son las siguientes:

- Planteado como un juego donde los usuarios van ganando puntos de experiencia conforme van aprendiendo el lenguaje objetivo. También incluye "vidas" para cada una de las lecciones, de manera que si el usuario se queda sin vidas antes de acabar la lección se ve forzado a reintentarla desde el principio.
- Incluye practicas con tiempo, donde por cada respuesta acertada se va ganando más tiempo para las siguiente preguntas.
- El sistema guarda información sobre qué preguntas le han resultado más complicadas al usuario y qué tipo de errores ha cometido, y aprovecha esta información de múltiples maneras.
- Recientemente se ha añadido un sistema de certificación mediante tests online.

Al igual que nuestra herramienta o los jueces on-line, Doulingo permite comprobar de manera inmediata si la solución introducida en el sistema es o no correcta. Existen estudios [40] que demuestran la eficacia del método Doulingo como método de aprendizaje de idiomas. Además, su madurez y presentación nos han servido como fuente de inspiración en este proyecto.

Una herramienta o conjunto de herramientas similar a Duolingo es Busuu [8]. Concebida como una red social (de hecho es la más grande en aprendizaje de idiomas), los usuarios hacen tanto de alumno (de la lengua o lenguas que quieren aprender) como tutor (de su lengua materna). Esto se alejaría del concepto de respuesta inmediata que perseguimos en este proyecto, pero aún así es interesante estudiarlas para ampliaciones futuras donde no pueda haber un sistema de corrección automático y se requiera la colaboración de una o más personas. Herramientas y páginas similares a Busuu son hello-hello [17], myngle [33], lingq [26], livemocha [29], babbel [5], Lang-8 [23] y Lingua.ly [27].

Otras formas de aprender nuevos lenguajes de programación partiendo de algún otro lenguaje incluye el aprendizaje por medio de crestomatías. Existen numerosos recursos en esta línea, donde tal

vez el que más destaque es Rosetta Code [38], una wiki de crestomatía de lenguajes de programación. Se basa en algoritmos comunes (unos 600) que son implementados en algunos de los diferentes lenguajes con los que cuenta el sistema (unos 500). Al tratarse de una wiki abierta a la colaboración está en constante crecimiento. La función principal de esta wiki es ilustrar cómo la misma funcionalidad puede ser implementada de manera muy diferente en cada paradigma o lenguaje de la programación. Puede ser muy útil para aprender un nuevo lenguaje mediante tareas que estén implementadas en el lenguaje objetivo y algún lenguaje que ya conozca el usuario en cuestión. La página langref.org [24] sigue un patrón muy parecido al de Rosetta Code.

6.2. Propuesta de diseño

Este proyecto consta de dos partes bien diferenciadas: la lógica, que se encarga de comprobar la corrección de un resultado introducido por el usuario con respecto a la solución previamente almacenada; y la interfaz gráfica, que presenta al usuario los problemas y su progreso y se encarga de interactuar con la lógica para calcular la adecuación de las respuestas del usuario.

6.2.1. Propuesta de diseño para la lógica

Los pasos a seguir para comprobar el grado de similitud entre dos programas son los siguientes:

1. El programa propuesto como solución debe ser analizado sintácticamente [2], para comprobar que se ajusta a la sintaxis del lenguaje y puede ser compilado y ejecutado. Este análisis generará un *árbol de sintaxis abstracta* que nos indicará de manera general cómo está estructurado el programa.

   Dado que este árbol es el punto de partida para la integración de nuevos lenguajes de programación, es importante que se haga de manera escalable para permitir la rápida integración de distintos lenguajes. Para ello, proponemos usar una herramienta que permita la generación automática de dicho árbol a partir de la gramática del lenguaje. De esta manera podremos trabajar con una estructura de datos común en todos los lenguajes, centrándonos en cada caso en las particularidades para el diagrama de flujo. En concreto, proponemos usar la herramienta ANTLR [35], que cumple estas condiciones y cuenta además con una biblioteca de gramáticas de lenguajes de programación que incluye Java y Python, lenguajes propuestos como prueba de concepto de nuestra herramienta.

2. A partir de dicho árbol de sintaxis abstracta generaremos un *diagrama de control de flujo* [3]. Este diagrama contendrá, por un lado, información general sobre los caminos que pueden ser usados por un programa durante su ejecución y, por otro lado, información específica sobre la estructura de control (e.g. bucle `while` o `for`). Esta información específica es relevante porque en nuestro sistema no estamos solo interesados en comparar el flujo de información, también resulta interesante que lo hagan con la misma estructura. Este requisito evita que un usuario aprenda solo ciertas estructuras de control y las use para resolver todos los problemas.

3. Una versión inicial del comparador podría comparar la similitud estricta entre nodos y aristas del diagrama de flujo de la solución y el diagrama correspondiente a la solución aportada por el usuario. La nota final se calcula en función de las similitudes entre ambos grafos.

4. Una mejora sencilla en el comparador consiste en permitir renombramientos de variables, que no afectan ni a la estructura ni al funcionamiento. Para ello, el comprobador debe calcular los posibles renombramientos y comprobar la igualdad en los términos anteriores tras aplicar cada posible sustitución.

5. La siguiente mejora consiste en calcular equivalencias entre condiciones, permitiendo así expresar condiciones de distintas maneras.

6. Una extensión de esta mejora es calcular equivalencias entre distintos caminos de ejecución, lo cual es especialmente interesante con instrucciones condicionales. Por ejemplo, podemos considerar que dos programas son equivalentes si el primero usa cierta condición y el segundo su negación, siempre y cuando las ramas `then` y `else` se intercambien.

7. La igualdad estricta se puede flexibilizar ejecutando fragmentos de programa y comprobando que ambos programas tienen los mismos efectos. Para ello, es recomendable usar mecanismos de seguridad como el *aislamiento de procesos* (*sandbox*) [4], para evitar que un usuario malintencionado pueda ejecutar código dañino para el servidor.

8. Por último, es interesante estudiar la posibilidad de usar ejecución simbólica [9] para comprobar la similitud entre bloques de código. La ejecución simbólica proporciona un mayor nivel de precisión a la hora de calcular la semejanza, pues condensa en una variable distintas clases de equivalencia de valores concretos. De esta manera, podemos evitar repeticiones y recorrer más caminos útiles.

6.2.2. Propuesta de diseño para la interfaz

Nuestro diseño para la interfaz de usuario consiste en un servicio web basado en una arquitectura RESTful [16]. Los recursos necesarios en esta arquitectura son:

Temas, a cargo de gestionar los distintos temas que pueden ser tratados. Un tema agrupa una cantidad amplia de problemas relacionados aunque probablemente con distintos niveles de dificultad y con distintas estructuras de control. Cada uno de estos temas estará compuesto por *lecciones*, como explicamos a continuación

Lecciones, que presentan un conjunto de problemas con una temática común, un nivel de dificultad similar y usando siempre las mismas estructuras de control. Cada tema está compuesto por distintos *ejercicios*.

Ejercicios, que consisten en un conjunto de enunciados, cada uno de ellos en un lenguaje de programación.

Enunciados, que implementan un programa en un lenguaje de programación concreto.

Lenguajes, que gestionan los distintos lenguajes de programación disponibles en la herramienta.

Candidatos, que se encargan de almacenar la información sobre posibles soluciones aportadas por los usuarios. Más información sobre los candidatos se proporciona en las siguientes secciones.

Usuarios, a cargo de la gestión de usuarios, lo que incluye el control de las lecciones que cada usuario ha terminado, sus favoritos y sus envíos. Para facilitar la gestión de usuarios se recomienda usar una aplicación externa.

Envíos, que almacenan la información acerca de las soluciones aportadas por los usuarios para cada ejercicio que ha realizado. Consideramos que esta información puede ser interesante para futuros análisis sobre el uso de la herramientas y las dificultades encontradas en cada ejercicio.

Asimismo, estos recursos dan lugar a una base de datos en la que tendremos una entidad por cada uno de los recursos arriba listados. Estos recursos debe interactuar con una interfaz que muestre al usuario su progreso hasta el momento y los candidatos que ha propuesto, así como las lecciones que están disponibles para su realización. Hemos decidido que la primera versión de la herramienta disponga de una interfaz web, dado que los ordenadores personales son todavía el principal medio de conexión a internet. Es interesante notar que este diseño tiene en cuenta dos puntos importantes especificados en la propuesta de proyecto:

- Facilita la integración con sistemas móviles, ya que el servicio web distingue entre el servidor, a cargo de la gestión de las bases de datos, y la interfaz, a cargo de la visualización. Aunque para un primer diseño hemos resuelto que sea web, esta puede ser sustituida por una interfaz móvil sin necesidad de cambiar la implementación de la parte del servidor.
- Incluye la integración con redes sociales, ya que propone identificar al usuario usando una cuenta previamente registrada en distintas redes sociales, lo que simplifica el almacenamiento de datos y permite el uso de las características disponibles en cada red social en concreto.

6.3. Temas en los cuales se puede dividir el aprendizaje

Los siguientes temas cubren un subconjunto suficientemente amplio de cualquier lenguaje de programación:

Expresiones. El tema *expresiones* permite a los usuarios conocer los distintos tipos básicos disponibles en el lenguaje, las operaciones disponibles sobre ellos y las instrucciones básicas como la asignación y la composición de instrucciones básicas. También se presentarán los tipos de datos necesarios para poder entender el resto de temas, como pueden ser las listas en el caso de Python. Este tema está accesible para el usuario desde el principio.

Funciones. El tema *funciones* introduce la sintaxis de las funciones, la forma de invocarlas, sus posibles modificadores de privacidad, las diferentes semánticas para los parámetros y las diferencias entre funciones y procedimientos. Este tema está disponible una vez el tema *expresiones* ha sido finalizado, aunque es posible que ciertas lecciones solo estén disponibles una vez se hayan completado lecciones posteriores. Es importante notar que las funciones son bloques básicos de los lenguajes de programación, por lo que se usarán en los siguientes temas y su clasificación dependerá del cuerpo de la función.

Condicionales. El tema *condicionales* presenta las distintas instrucciones de control que permitan establecer distintos caminos de ejecución, como son las instrucciones `if`, `case` o `switch`. Este tema estará disponible una vez se hayan completado las lecciones básicas del tema *funciones*.

Bucles. El tema *bucles* presenta la sintaxis de los bucles, e.g. `while` y `for`. Dado que este tema presenta en general mayor complejidad que los condicionales, consideramos que aquellos ejercicios que combinen ambas estructuras se encuadran en el tema de bucles; por ello, es necesario haber superado el tema *condicionales* para acceder a este tema.

Recursión. El tema *recursión* presenta los distintos esquemas para programar y optimizar funciones recursivas. Este tema estará disponible una vez se haya completado el tema *condicionales*.

Clases. El tema *clases* presenta cómo programar una clase, haciendo énfasis en las ideas básicas de encapsulamiento, herencia y polimorfismo, y describiendo las principales características de los atributos, constantes y funciones.

Estructuras de datos. El tema *estructuras de datos* permite a los usuarios definir estructuras de datos básicas y operaciones sobre ellas. Los problemas de referencia para este tema serán los descritos en [31].

Métodos algorítmicos. El tema *métodos algorítmicos* presenta los principales esquemas algorítmicos para resolución de problemas. Los problemas de referencia para este tema serán de nuevo los presentados en [31].

Es importante hacer notar que ciertos módulos pueden no adecuarse a un cierto lenguaje. Por ejemplo, una extensión de la herramienta para lenguajes declarativos no usaría el módulo dedicado a bucles. Es también posible que alguna de las características en las descripciones de los módulos

no estén disponibles en un lenguaje particular. En tales casos, dichos ejercicios se omitirán, aunque el resto del módulo seguirá siendo válido. En la siguiente sección entramos en detalles sobre estas divisiones.

6.3.1. Lecciones necesarias en cada módulo

Como hemos indicado, consideramos que cada tema se divide en una serie de lecciones. Dichas lecciones pueden separar, bien conceptos diferentes dentro de un mismo tema (como, por ejemplo, los bucles `while` y `for` dentro del tema *bucles*), bien diferentes niveles de dificultad para un mismo concepto (como, por ejemplo, bucles vs. bucles anidados).

Desde un punto de vista didáctico está división es interesante porque permite al estudiante centrarse en un contenido específico y con una dificultad que él considera adecuada. Además, es también importante desde el punto de vista de la usabilidad, pues permite realizar lecciones cortas y directas al interés del usuario.

También es interesante notar que en algunos casos podemos tener más ejercicios en una tarea de los que la herramienta muestra en una sesión normal. Esto permite que distintas ejecuciones de la tarea presenten diferentes problemas, evitando así que el usuario busque finalizar la tarea por simple memorización, sino que debe aprender los conceptos y aplicarlos adecuadamente.

Así pues, consideramos que una posible distribución de los ejercicios es la siguiente:

- Tema *expresiones*:
  - Expresiones númericas, haciendo especial énfasis en las cambios de tipo al mezclar diferentes números (e.g. enteros y *float*), incluyendo operadores de redondeo y la división entera.
  - Booleanos, incluyendo tanto operaciones lógicas como operaciones de comparación entre otros tipos.
  - Cadenas, incluyendo funciones de búsqueda de subcadenas e impresión por pantalla.
  - Listas/arrays, incluyendo acceso, inserción y borrado.
- Tema *funciones*:
  - Usos básicos: cálculo de expresiones simples.
  - Invocación de funciones desde otras funciones.
  - Semánticas en el paso de parámetros.
  - Procedimientos vs. funciones.
- Tema *condicionales*:
  - Condicionales simples.
  - Condicionales anidados.
- Tema *bucles*:
  - Bucles simples.
  - Optimizaciones en bucles: mejorando las condiciones.
  - Combinación de bucles y condicionales.
  - Bucles anidados.
- Tema *recursión*:
  - Recursión básica.
  - Recursión múltiple.

  - Recursión mutua.
  - Técnicas de inmersión para funciones recursivas.
  - Transformación de recusión no final en recursión final.

- Tema *clases*:
  - Sintaxis clases.
  - Encapsulamiento.
  - Herencia.
  - Polimorfismo.

- Tema *estructuras de datos*:
  - Estructuras lineales: pilas, colas y listas.
  - Árboles: árboles binarios, árboles generales, árboles de búsqueda y árboles equilibrados.
  - Grafos.

- Tema *métodos algorítmicos*:
  - Divide y vencerás, incluyendo algoritmos de ordenación.
  - Método voraz, incluyendo algoritmos sobre grafos.
  - Programación dinámica, distinguiendo su dificultad según el número de parámetros necesarios en la fórmula que soluciona el problema y la complejidad de la reducción de espacio.
  - Vuelta atrás, prestando atención a posibles técnicas de optimización.
  - Ramificación y poda, prestando atención al esquema optimista-pesimista.

6.4. Capacidad de mejora mediante la interacción entre los usuarios

Consideramos que en este proyecto se deben distinguir dos categorías:

- Los usuarios básicos, que pueden:
  - Proponer nuevas soluciones para problemas ya resueltos. Cuando nos enfrentamos a problemas de programación es frecuente que un mismo problema tenga distintas soluciones, todas ellas igual de correctas. Dado que nuestra herramienta inicialmente solo tendrá una posible solución para cada problema, es probable que los usuarios desarrollen nuevas soluciones válidas para el problema. En tal caso dichos usuarios podrán proponer su solución como *candidata* que será posteriormente evaluada, como veremos a continuación.
  - "Juzgar" las soluciones propuestas por otros usuarios. Es habitual en los sistemas colaborativos el tener un filtro previo para evaluar la adecuación de las propuestas hechas por los usuarios antes de que dichas propuestas sean evaluadas por los administradores. En nuestro caso proponemos que los usuarios puedan ver los candidatos propuestos por otros usuarios y juzgar si es o no una solución válida. Si cierto candidato recibe una cierta cantidad de votos positivos entonces será elevado a un administrador para que decida si dicho candidato se añade definitivamente al conjunto de soluciones.

- Los administradores, que pueden:
  - Tomar la decisión final sobre un candidato. Como hemos visto en el punto anterior, los candidatos reciben votos por parte de los usuarios y, una vez alcanzan un cierto número de votos positivos (respectivamente, negativos) un administrador decide si añadir el candidato al conjunto de soluciones posibles (respectivamente, eliminarla como candidato).

- Añadir nuevos problemas. Si un administrador decide que un ejercicio nuevo puede ser interesante, deberá ubicarlo en el módulo y tarea correspondientes (los módulos y las tareas se describen en las siguientes secciones). Una vez añadido el problema, será necesario añadir soluciones en todos los lenguajes.
- Añadir soluciones. Como se ha indicado en el punto anterior, es posible que se añadan nuevos problemas, por lo que es necesario
- Añadir nuevos lenguajes. En este caso se crea un nuevo "hueco" para que todos los problemas ya definidos para el resto de lenguajes se pueda resolver en el nuevo lenguaje.

6.5. Posibilidad de expedir certificados

El objetivo final de esta herramienta es enseñar a los estudiantes cómo programar en un cierto lenguaje y, por tanto, al finalizar con éxito el curso sería posible expedir un certificado. Como en la mayor parte de las plataformas para aprendizaje en línea (e.g. edX [15] o Coursera [10]), es necesario distinguir en este punto los certificados basados en el "honor", en los que el alumno se compromete a no hacer trampas para completarlo y que por lo tanto ofrecen un nivel de seguridad bajo, y los certificados "validados", en los que los alumnos prueban su identidad antes de realizar tareas puntuables. En una primera fase proponemos únicamente generar certificados basados en el honor, dado que los certificados validados requieren recursos fuera del alcance del presente estudio.

Por otro lado, en la actualidad ha cobrado interés el uso de "insignias", que acreditan normalmente conocimientos en un área específica. Estas insignias se usan para acreditar méritos en plataformas de búsqueda de empleo online como linkedIn [28], y por ello diversas plataformas, como Mozilla OpenBadges [34] y Credly [11], se han creado para permitir su almacenamiento y distribución. Por otro lado, también han surgido herramientas para facilitar su creación, como BadgeOS [6]. Esta tendencia nos lleva a proponer la expedición de insignias cuando un usuario supere módulos avanzados o colabore en actividades específicas. Igual que en el caso anterior, sería posible distinguir entre las insignias obtenidas por usuarios validados y por aquellos basados en el honor.

6.6. Integración con el Campus Virtual

El diseño del sistema expuesto anteriormente proponía una separación del sistema en dos partes: la lógica interna, que comprueba la validez de la respuestas introducidas por el estudiante, y la parte frontal, que es aquella con la que el usuario interactúa directamente. Esta separación, junto con la decisión de diseñar la interfaz mediante un servicio web *RESTful*, facilita en gran medida la integración de nuestro entorno con los sistemas CMS (*Course Management Systems*), tales como *Moodle* [32], *Sakai* [39] o *Blackboard* [7], los dos primeros integrados en el Campus Virtual de la UCM.

Ninguna de las herramientas actualmente proporcionadas por *Moodle* o *Sakai* puede utilizarse directamente para nuestros propósitos. Ambas plataformas disponen de un módulo de cuestionarios que permiten al profesor introducir una batería de preguntas de distintos tipos: opción simple, opción múltiple, respuesta corta, respuesta numérica, emparejamientos, etc. El estudiante, al realizar el cuestionario, ha de contestar correctamente a una selección aleatoria tomada a partir de esta batería de preguntas. Sin embargo, los métodos de corrección automática utilizados por este tipo de herramientas son demasiado simples para nuestro propósito. Por ejemplo, en aquellas preguntas de respuesta corta, el profesor introduce a mano una lista de posibles respuestas válidas, y el sistema considera que la respuesta de un estudiante es correcta si coincide literalmente con alguna de estas respuestas. Sin embargo, nuestro sistema requiere realizar una llamada a un servicio web externo, que es el que implementa la lógica interna de la aplicación para asignar una puntuación a la respuesta introducida por el usuario.

De lo anterior se desprende que es necesario el desarrollo de un añadido (*plug-in*) para cada una de estas plataformas. *Moodle* proporciona una lista de categorías de añadidos, que no son más que plantillas que el desarrollador ha de completar para personalizar el comportamiento de un determinado

módulo en el curso virtual. Por ejemplo, disponemos de plantillas para desarrollar generadores de informes de calificaciones, tipos alternativos de calendarios, exportadores de listados de calificaciones a otros formatos, etc. A la hora de desarrollar un añadido en esta plataforma es recomendable basarse en una de estas categorías antes que hacerlo desde cero, ya que facilita no solo su desarrollo, sino su mantenibilidad a lo largo de las distintas versiones de *Moodle*.

Nuestro entorno de aprendizaje colaborativo requiere el desarrollo de añadidos a partir de las siguientes categorías, basadas en el motor de preguntas de *Moodle*:

- *Question types*. Extendiendo esta categoría podemos controlar la representación de la pregunta.
- *Question behaviour*. Sirve para especificar la interacción del estudiante con la pregunta en cada intento y la corrección de la respuesta introducida por el mismo. La extensión debe contemplar la posibilidad, antes comentada, de que el estudiante proponga su solución como candidata a ser evaluada como correcta por un administrador.

Otra ventaja importante de plantear el añadido de *Moodle* a partir de extensiones de categorías es que la gestión de temas, lecciones, ejercicios, envíos, etc. puede realizarse utilizando las herramientas ya existentes de esta plataforma. En particular, al plantear cada lección (entendida como un conjunto de ejercicios) como un cuestionario *Moodle*, la gestión de calificaciones ya viene proporcionada por esta plataforma. El módulo de cuestionarios ya es lo suficientemente flexible como para permitir al estudiante enviar cada pregunta por separado para cada su corrección sin necesidad de haber respondido a todas ellas.

Además del motor de preguntas mencionado anteriormente existe otro componente esencial de *Moodle* candidato a ser integrado con nuestro sistema: el módulo de lecciones. Con ello se pretende contemplar la posibilidad de aprendizaje de un lenguaje de programación a partir de cero, es decir, sin requerir conocimientos de otro lenguaje de programación distinto, tal y como se comentará en la sección siguiente. El módulo de lecciones permite al sistema intercalar explicaciones en lenguaje natural con cuestionarios en los que el estudiante ejercita lo aprendido mediante la traducción de pseudo-código.

La integración de nuestro sistema de aprendizaje con la plataforma *Sakai* también es viable. La gran ventaja de esta plataforma con respecto a *Moodle* está en el hecho de que la implementación de *Sakai* se apoya en una serie de tecnologías y marcos de desarrollo Java (*Spring*, *Hibernate*, etc.) que son de muy amplio uso en las aplicaciones web basadas en este lenguaje. Esto facilita la tarea del desarrollo de extensiones (conocidas en *Sakai* como *aplicaciones*) por parte de programadores expertos en dichas tecnologías. Sin embargo, y a diferencia de la plataforma *Moodle*, aquí no disponemos de un conjunto de añadidos básicos a partir de los cuales desarrollar nuestras propias extensiones, si no que se tendrían que desarrollar desde cero. En este sentido consideramos que *Sakai* es menos modular que *Moodle*.

6.7. Posibilidad de aprendizaje a partir de cero

Nuestro diseño contempla la posibilidad de aprender un lenguaje de programación sin necesidad de tener conocimientos previos de otros lenguajes. En tal caso, el lenguaje fuente sería pseudo-código o lenguaje natural, mientras que el lenguaje destino sería el lenguaje que se desea aprender. Esta opción permite aprender todos los lenguajes ya integrados en el sistema utilizando como “enunciado” estos problemas. En el caso del pseudo-código la tarea consistiría en adaptar el algoritmo, ya estructurado, al lenguaje elegido, mientras que al elegir lenguaje natural el usuario no solo necesita conocer bien la sintaxis del lenguaje, sino que también es necesario entender cómo estructurar funciones, por lo que podrían necesitarse explicaciones adicionales. Este último aspecto es también una importante desventaja desde el punto de vista de la implementación, ya que el número de posibles soluciones se eleva de manera notoria. Por ello, sería necesario modificar el comparador para permitir comparaciones más flexibles.

Una restricción importante aplicada a estos lenguajes es que no pueden seleccionarse como lenguaje destino, dado que estos ejercicios no proporcionarían a los usuarios ninguna competencia en lenguajes de programación.

6.8. Posibilidad de añadir nuevos lenguajes de programación

La metodología a seguir para introducir un nuevo lenguaje de programación en el sistema es la siguiente:

1. Como se ha explicado en apartados anteriores, la comparación entre respuestas está basadas en diagramas de flujo. Este diagrama se obtiene a partir de un árbol de sintaxis abstracta generado automáticamente a partir de la gramática del lenguaje a añadir. Por ello, el primer paso consiste en definir la gramática del lenguaje de programación que deseamos añadir.

2. Una vez obtenido el árbol de sintaxis abstracta, se debe generar un diagrama de flujo. La estructura del algoritmo es similar para todos los lenguajes, pues solo varían los nombres concretos de los símbolos en el lenguaje. Por ello, este proceso puede completarse en un tiempo razonablemente corto.

3. Una vez obtenido el diagrama de flujo, se pueden usar los algoritmos de comparación de diagramas ya implementados para las comparaciones.

4. Por otra parte, es necesario añadir las soluciones a todos los problemas propuestos en el nuevo lenguaje. Para ello es necesario que un experto supervise la corrección de dichas soluciones antes de añadirlas a la base de datos, ya que servirán de modelo para las correcciones.

6.9. Posibilidades de uso del proyecto fuera del ámbito académico

Un rápido repaso a las ofertas de empleo en Informática que se encuentran disponibles en la actualidad solicitan programadores para migrar sistemas que han quedado ya obsoletos a nuevas plataformas. El sistema propuesto en esta memoria permitiría realizar estas migraciones añadiendo una nueva categoría en la que usuarios avanzados, es decir, aquellos que ya han superado los módulos estándar, pueden trabajar en la traducción de parte del sistema. De esta manera, mediante las traducciones conjuntas de diversos usuarios se obtendría la traducción del sistema completo. Es interesante notar que esta traducción permitiría mejorar, de ser necesario, la modularidad del sistema si se incorpora un paso previo en la que un experto (es decir, uno de los administradores) rediseña el software.

Esta función sería similar a la "inmersión" ofrecida por el sistema Duolingo[2], en la que los usuarios traducen textos reales por cuya traducción Google recibe un pago. Estas traducciones permiten elegir pequeños textos (previamente delimitados) y, bien traducirlos, bien juzgar la traducción. Una vez un texto ha alcanzado cierta puntuación se considera correcto y otras partes del texto deben ser traducidas. Es interesante notar que este mecanismo ya está contemplado para nuestro sistema para juzgar los candidatos a solución propuestos por un usuario.

Por supuesto, medidas adicionales de seguridad deberían tenerse en cuenta a la hora de realizar trabajos de este tipo, ya que los sistemas software presentan mayor riesgo que la simple traducción de textos. Además de solicitar el visto bueno de diversos usuarios, que proporciona un nivel mínimo de seguridad, sería necesario tener un conjunto suficientemente completo de casos de prueba que permita comprobar que el sistema sigue presentando el mismo comportamiento.

6.10. Requisitos de personal

Consideramos que cada una de las partes indicadas en el punto anterior (implementación de la lógica y de la interfaz) puede llevarse a cabo por estudiantes de Informática como Trabajo de Fin de

[2] `http://duolingo.wikia.com/wiki/Immersion`

Grado. Asimismo, es necesario desarrollar el conjunto de soluciones que serán usadas para comprobar la corrección de las respuestas del usuario. Consideramos que esta tarea también puede ser interesante para la formación en programación de estudiantes de último año, especialmente de la Facultad de Matemáticas, donde la carga de programación es menor que en la Facultad de Informática. Estas soluciones contendrán tanto el código que soluciona el problema como un conjunto de casos de test probando que funciona correctamente.

Asimismo, las extensiones futuras (e.g. los propuestos en la sección 6.6) también podrán llevarse a cabo en posteriores Trabajos Fin de Grado, una vez la herramiente esté operativa y se haya demostrado su utilidad.

Referencias

[1] Acepta el reto homepage. `https://www.aceptaelreto.com/`.

[2] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[3] F. E. Allen. Control flow analysis. *ACM SIGPLAN Notices*, 5(7):1–19, July 1970.

[4] J. Ansel, P. Marchenko, Úlfar Erlingsson, E. Taylor, B. Chen, D. L. Schuff, D. Sehr, C. L. Biffle, and B. Yee. Language-independent sandboxing of just-in-time compilation and self-modifying code. *ACM SIGPLAN Notices - PLDI 2011*, 46(6):355–366, June 2011.

[5] babbel homepage. `http://es.babbel.com/`.

[6] BadgeOS homepage. `http://badgeos.org/badgestack/`.

[7] Blackboard homepage. `http://www.blackboard.com/Platforms/Learn/Overview.aspx/`.

[8] Busuu homepage. `https://www.busuu.com/`.

[9] C. Cadar and K. Sen. Symbolic execution for software testing: Three decades later. *Communications of the ACM*, 56(2):82–90, Feb. 2013.

[10] Coursera homepage. `https://www.coursera.org/`.

[11] Credly homepage. `https://credly.com/`.

[12] C. Douce, D. Livingstone, and J. Orwell. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.*, 5(3), Sept. 2005.

[13] Duolingo homepage. `https://www.duolingo.com/`.

[14] S. H. Edwards. Improving student performance by evaluating how well students test their own programs. *J. Educ. Resour. Comput.*, 3(3), Sept. 2003.

[15] edX homepage. `https://www.edx.org/`.

[16] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, May 2002.

[17] hello-hello homepage. `http://www.hello-hello.com/`.

[18] C. Higgins, T. Hegazy, P. Symeonidis, and A. Tsintsifas. The coursemarker cba system: Improvements over ceilidh. *Education and Information Technologies*, 8(3):287–304, Sept. 2003.

[19] C. A. Higgins, G. Gray, P. Symeonidis, and A. Tsintsifas. Automated assessment and experiences of teaching programming. *J. Educ. Resour. Comput.*, 5(3), Sept. 2005.

[20] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 86–93, New York, NY, USA, 2010. ACM.

[21] M. Joy, N. Griffiths, and R. Boyatt. The boss online submission and assessment system. *J. Educ. Resour. Comput.*, 5(3), Sept. 2005.

[22] R. Kolstad. Infrastructure for contest task development. *Olympiads in Informatics*, 3:38–59, 2009.

[23] Lang-8 homepage. http://lang-8.com/.

[24] langref.org homepage. http://langref.org/.

[25] J. P. Leal and F. Silva. Mooshak: A web-based multi-site programming contest system. *Softw. Pract. Exper.*, 33(6):567–581, May 2003.

[26] lingq homepage. http://www.lingq.com/.

[27] Lingua.ly homepage. http://lingua.ly/.

[28] LinkedIn homepage. https://www.linkedin.com/.

[29] livemocha homepage. http://livemocha.com/.

[30] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti. Visual algorithm simulation exercise system with automatic assessment: Trakla2. *Informatics in Education*, 3(2):267–288, 2004.

[31] N. Martí-Oliet, Y. Ortega, and A. Verdejo. *Estructuras de datos y métodos algorítmicos (Ejercicios resueltos)*. Pearson Educación, 2004.

[32] Moodle homepage. https://moodle.org/.

[33] myngle homepage. http://www.myngle.com/.

[34] Mozilla OpenBadges homepage. http://openbadges.org/.

[35] T. Parr. *The Definitive ANTLR 4 Reference*. The Pragmatic Programmers, 2nd edition, 2013.

[36] J. Petit, O. Giménez, and S. Roura. Jutge.org: An educational programming judge. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 445–450, New York, NY, USA, 2012. ACM.

[37] M. Revilla, S. Manzoor, and R. Liu. Competitive learning in informatics: The UVa online judge experience. *Olympiads in Informatics*, 2:131–148, 2008.

[38] Rosetta code homepage. http://rosettacode.org/.

[39] Sakai homepage. https://sakaiproject.org/.

[40] R. Vesselinov and J. Grego. Duolingo effectiveness study. *City University of New York, USA*, 2012.