UNIVERSIDAD
COMPLUTENSE
MADRID

Proyecto de Innovación

Convocatoria 2018/19

Proyecto nº 61

Herramienta interactiva de manipulación y corrección de entregas de ejercicios en asignaturas con evaluación continua

Manuel Montenegro Montes

Facultad de Informática

Universidad Complutense de Madrid

1. Objetivos propuestos en la presentación del proyecto

Este proyecto tiene como objetivo extender el conjunto de herramientas realizadas en un proyecto de innovación anterior: *Validación en línea de aspectos formales y corrección asistida de ejercicios en asignaturas con evaluación continua* (Innova-Docencia 2016, nº 49) [MSV+17]. Este último estaba dirigido a aquellas asignaturas con mecanismos de evaluación continua, los cuales, pese a sus múltiples ventajas para el aprendizaje por parte de los/las estudiantes, resultan poco operativos en aquellas asignaturas con grupos numerosos debido a la gran cantidad de esfuerzo que conlleva corregir todas las entregas de los ejercicios en un tiempo razonable. Esto limita en gran medida el margen de maniobra del profesor/a a la hora de proponer ejercicios y prácticas, bien viéndose en la obligación de proponer ejercicios de reducida envergadura (con el fin de corregirlos rápidamente), o bien limitando el número de ejercicios de evaluación continua a realizar durante el curso.

Aunque la corrección de muchos de estos ejercicios se realice de manera manual, gran parte de los procesos involucrados en esta corrección son automatizables. Entre estos procesos podemos citar: compilación de programas, ejecución de casos de prueba, conversión de documentos de un formato a otro, extracción de información relevante para la evaluación, etc. Para ello, un profesor puede escribir un programa de procesamiento por lotes (*script*) que realice automáticamente este tipo de tareas. Esto requiere, por un lado, disponer de ciertos conocimientos de programación y, por otro lado, requiere que el profesor anticipe de antemano los resultados de cada una de las acciones del mismo, incluyendo la gestión de los posibles errores que pudieran producirse. Con el presente proyecto de innovación se pretende facilitar el procesamiento automático de las entregas de un ejercicio sin necesidad de que el profesor tenga que escribir un programa específico para ello. Para ello se presenta en este proyecto una herramienta que permita a un/a profesor/a explorar, de modo interactivo, las múltiples entregas de los estudiantes relativas a un mismo ejercicio, realizar acciones sobre ellas, y extraer de las mismas aquella información que sea relevante para la evaluación.

El objetivo del proyecto es, por tanto, el **diseño e implementación de un asistente interactivo de corrección de ejercicios y prácticas**. Este asistente permite procesar todas las entregas de un mismo ejercicio como si de una única entrega se tratase. En particular, se aborda el desarrollo de la siguiente funcionalidad:

- **Exploración de los componentes comunes a todas las entregas**, mediante una interfaz similar a la del explorador de ficheros de un entorno de escritorio. Por ejemplo, supongamos que la entrega de cada estudiante se compone de una serie de ficheros y carpetas. Una vez recolectadas todas las entregas de los estudiantes, el asistente muestra una vista con aquellos ficheros y carpetas comunes a todas las entregas. También se contempla la posibilidad de mostrar los ficheros y carpetas que solamente aparezcan en algunas de las entregas, y no en todas, pero estos últimos se mostrarán en segundo plano. De este modo, el sistema proporciona la apariencia de estar navegando por una única entrega “modelo”, cuando en realidad se están explorando simultáneamente las entregas de todos los estudiantes.

- **Procesamiento por lotes de los componentes de todas las entregas**. Utilizando la misma vista de explorador de ficheros, el/la profesor/a puede seleccionar uno de los componentes de la entrega con el fin de realizar una determinada operación sobre él. De nuevo, aunque el sistema proporciona la apariencia de estar realizando esta operación sobre una única componente modelo, en realidad esta operación se estará replicando en cada entrega individual de cada alumno. En el ejemplo anterior, el profesor podrá seleccionar uno de los ficheros comunes a todas las entregas y aplicar cualquiera de las operaciones disponibles: renombrar, mover a otra carpeta, eliminar, etc. Si el profesor renombra uno de los ficheros, el renombramiento del fichero se realizará sobre todas las entregas.

- **Extracción de los componentes relevantes para la evaluación.** Dentro de la misma interfaz de exploración, el/la profesor/a podrá seleccionar un componente específico de la entrega modelo, y el asistente recorrerá todas las instancias de ese componente en cada una de las entregas para agruparlas en un único fichero. Esta funcionalidad resulta útil cuando solamente algunos de los componentes de la entrega han de ser revisados manualmente. Por ejemplo, supongamos que, en una asignatura de Informática, los/as estudiantes han de entregar la implementación de un determinado algoritmo. Cada estudiante debe entregar, además del código fuente del algoritmo, una serie de ficheros adicionales con casos de prueba, código objeto, etc. Estos ficheros adicionales pueden ser procesados automáticamente, mientras que el código fuente ha de ser revisado manualmente por el docente. Para esto último, el docente puede agrupar los códigos fuentes de todas las entregas en un único fichero, con el fin de agilizar la corrección del mismo.

Cuanto mayor similitud formal haya entre las entregas de los distintos alumnos, más eficaz será la interacción del profesor con el asistente propuesto en este proyecto. Por tanto, en todo momento supondremos que las entregas gestionadas por el asistente de corrección han sido validadas previamente por las herramientas desarrolladas en el proyecto anterior [MSV+17] con respecto a una descripción formal de requisitos.

Asimismo, el sistema desarrollado es multiplataforma, con el fin de que pueda ser utilizado por el mayor número posible de usuarios. También se contempla en todo momento la extensibilidad del proyecto mediante añadidos (*plugins*), de modo que pueda ser aplicable sin dificultad a las distintas asignaturas.

2. Objetivos alcanzados

En esta sección detallaremos hasta qué punto se han conseguido los objetivos expuestos en la sección anterior. Teníamos como objetivo principal el desarrollo de una herramienta interactiva de corrección, cuya funcionalidad se desglosa en los siguientes subobjetivos:

Exploración de los componentes de las entregas

Este objetivo se ha completado en su totalidad. Una vez cargadas todas las entregas de un mismo ejercicio, el asistente permite navegar por los atributos y los ficheros de cada una de las entregas. Se han habilitado dos vistas de navegación.

- **Vista individual**. Mediante esta vista se puede observar el contenido de un determinado elemento, que puede formar parte de una o varias entregas. En este último caso se muestra el contenido de dicho elemento de forma separada para cada una de las entregas. Por ejemplo, supongamos que la entrega de un ejercicio consiste en el envío de varios ficheros contenidos en un mismo directorio, y que dicho ejercicio ha sido entregado por diez estudiantes, cada uno/a enviando su propio directorio. En la vista individual se mostraría un listado con las diez entregas, cada una conteniendo el contenido de los directorios (Figura 1).

- **Vista de grupo.** Al igual que la vista individual, permite observar el contenido de un determinado elemento. La diferencia con respecto a la anterior es que, en caso de que el elemento forme parte de varias entregas, se mostrará su contenido de forma agregada. En el ejemplo anterior, se mostraría un único listado conteniendo todos los ficheros de todas las entregas. Si un fichero tiene el mismo nombre en distintas entregas, aparece una única vez. Cada uno de los elementos del listado aparece un indicador del número de entregas que contienen ese fichero (Figura 2).

La vista de grupo solamente está disponible en aquellos elementos que admiten una representación multivalorada. Entre ellos se encuentran, por ejemplo, los directorios, pues contienen varios ficheros. Por el contrario, los elementos con tipos atómicos (cadenas de texto, números enteros, etc) solo admiten la vista individual.

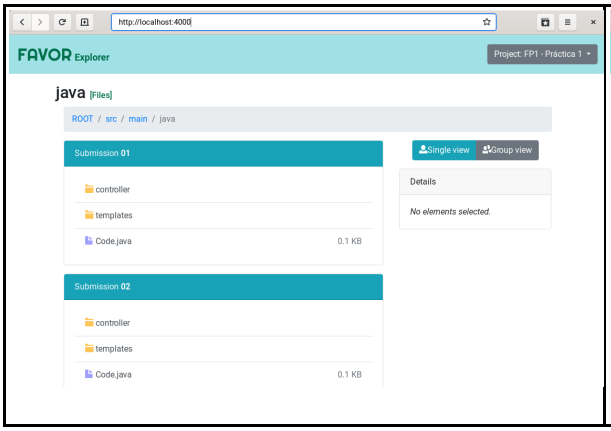

Figura 1. Vista individual

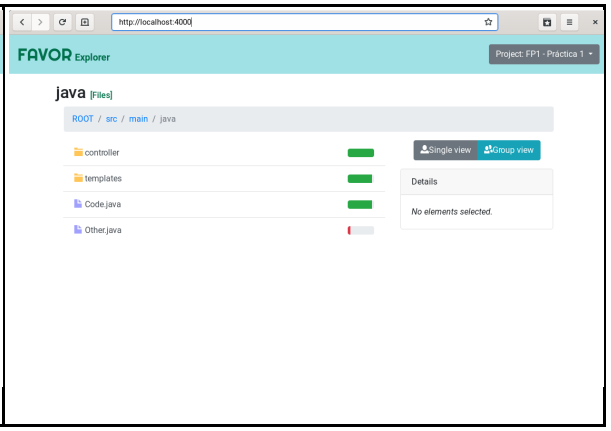

Figura 2. Vista de grupo

Procesamiento por lotes de las componentes de las entregas

Este objetivo también se ha completado. Cuando el/la usuario/a del asistente selecciona un elemento en alguna de las dos vistas (individual o grupo), se mostrará un cuadro con los detalles del elemento seleccionado. Esta vista de detalle contiene un conjunto de botones que permiten realizar acciones sobre el elemento seleccionado. En una primera fase se han implementado las siguientes operaciones:

- Renombrar fichero/directorio.
- Eliminar fichero/directorio.
- Ejecutar comando de *shell* en un fichero/directorio.

El hecho de haber escogido estas tres operaciones se debe a que son las más comunes durante el proceso de entregas de ejercicios. No obstante, la incorporación de nuevas acciones resulta relativamente sencilla, y se mantiene como una futura extensión del proyecto.

Extracción de los componentes relevantes para la evaluación

Aunque de una forma distinta a la inicialmente prevista, este objetivo también se ha conseguido. En un principio se planeó la funcionalidad de que el/la usuario/a del asistente seleccionase un determinado elemento de las entregas, y el sistema recorriese todas las entregas para obtener dicho elemento, para así agrupar los resultados obtenidos en un fichero. Sin embargo, una vez implementada la funcionalidad de explorar las componentes de las entregas en sus dos vertientes (vista individual y de grupo), se constató que no era necesario el desarrollo de esta funcionalidad, ya que la vista individual (Figura 1) realiza algo similar. En efecto, mediante dicha vista es posible obtener un resumen de todas las entregas en una misma página. Esta página puede exportarse a otros formatos (HTML o PDF) mediante las opciones de impresión del navegador.

Si el/la usuario/a del asistente desea obtener resúmenes más complejos (por ejemplo, que involucren la mezcla de varias componentes de una entrega), sería necesaria la implementación de una herramienta específica de informes en el asistente. Esto se plantea como una futura extensión del asistente.

3. Metodología empleada en el proyecto

El desarrollo se llevó a cabo de modo secuencial, ya que los tres objetivos mencionados en las secciones anteriores son incrementales. Para cada objetivo se realizó una tarea previa en el que se analizaba el modelo de datos necesario con el fin de que el asistente tratase los elementos de las entregas del modo más genérico posible. A continuación se muestran las tareas destinadas a la consecución de cada objetivo. La planificación temporal se muestra en la Figura 3.

Objetivo 1. Exploración de componentes.

- **Tarea 1.1.** Definición del modelo de visualización de los componentes de una práctica.
- **Tarea 1.2.** Selección de tecnologías para la implementación de la interfaz gráfica.
- **Tarea 1.3.** Implementación del explorador de componentes de una práctica.
- **Tarea 1.4.** Pruebas de ejecución.

Objetivo 2. Procesamiento por lotes de componentes.

- **Tarea 2.1.** Definición de modelo de acciones sobre entidades.
- **Tarea 2.2.** Extensión de la herramienta para incorporar la ejecución de acciones.
- **Tarea 2.3.** Pruebas de ejecución.

Objetivo 3. Extracción de componentes específicos.

- **Tarea 3.1.** Implementación de las acciones correspondientes a la extracción.
- **Tarea 3.2.** Pruebas de ejecución.

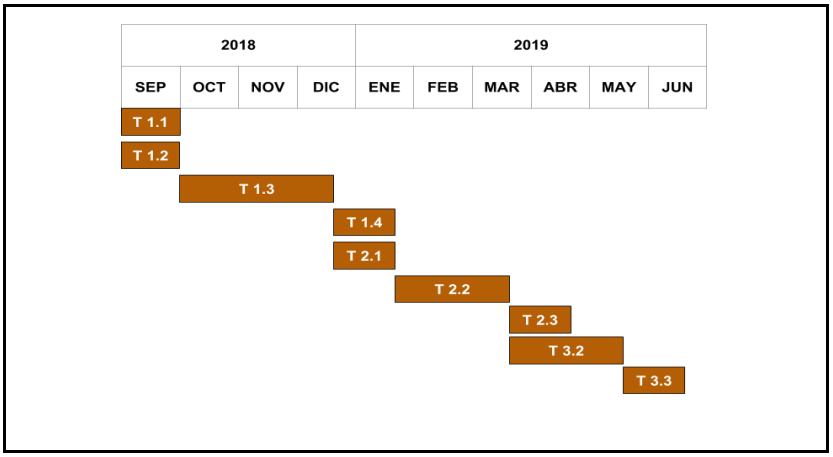

Figura 3. Planificación temporal

4. Recursos humanos

El equipo del proyecto está formado por tres profesores doctores (Enrique Martín, Manuel Montenegro y Adrián Riesco) y dos estudiantes de doctorado (Santiago Saavedra y Gorka Suárez), de los cuales uno (Gorka Suárez) ha trabajado como personal de apoyo a la investigación (PAI). En la siguiente tabla se resumen la participación de cada miembro del proyecto en la consecución de los objetivos del mismo.

	Enrique Martín	Manuel Montenegro	Adrián Riesco	Santiago Saavedra	Gorka Suárez
Tarea 1.1		X		X	
Tarea 1.2		X		X	
Tarea 1.3		X			X
Tarea 1.4	X		X		
Tarea 2.1				X	
Tarea 2.2		X			X
Tarea 2.3	X		X		
Tarea 3.1		X			X
Tarea 3.2	X		X		

5. Desarrollo de las actividades

De las tareas mostradas en las secciones anteriores, se desarrollan en esta sección las relativas a la selección de tecnologías y a la descripción de los modelos de propiedades y acciones. Los resultados relativos a implementación y testeo pueden encontrarse directamente en el depósito *GitHub* del proyecto.

Selección de tecnologías

Una de las tareas realizadas al inicio del proyecto consiste en el estudio de las tecnologías adecuadas para abordar el desarrollo de la aplicación. Para ello se establecieron previamente una serie de propiedades deseables en el sistema desarrollado, denominadas *requisitos no funcionales*, ya que involucran aspectos que van más allá de la funcionalidad percibida por el usuario (esto es, los objetivos expuestos en la sección 1). Estos requisitos son los siguientes:

- **Multiplataforma.** Existe una amplia heterogeneidad en el conjunto de sistemas operativos y plataformas utilizados por el personal docente e investigador de la Facultad de Informática. Si el asistente funcionase en un único sistema operativo (por ejemplo, Windows) su aceptación por parte del PDI se vería notablemente afectada. Las tecnologías utilizadas en el desarrollo deben permitir el uso del asistente, al menos, en sistemas Windows, macOS, y aquellos basados en Linux.

- **Carga dinámica de módulos**. En el proyecto de innovación anterior [MSV+17] se puso especial énfasis en poder acomodar distintos tipos de elementos dentro de una entrega con el fin de poder fijar una amplia variedad de restricciones sobre los mismos, pudiendo extender posteriormente la cantidad de elementos soportados mediante añadidos (*plugins*). Con el presente proyecto se desea continuar la misma línea de extensibilidad basada en *plugins*. Por este motivo, buscamos un lenguaje y un entorno de ejecución que permitan la carga dinámica de módulos sin tener que recompilar el asistente cada vez que se incorpore un nuevo módulo.

- **Arquitectura web.** Aunque el asistente de corrección está inicialmente concebido como una aplicación de escritorio, se prevé en un futuro su uso como una aplicación web. Por tanto, se valoran aquellas tecnologías y lenguajes de programación integrables en una arquitectura cliente-servidor, y establezcan fácilmente una separación entre la interfaz gráfica de usuario y la lógica de la aplicación. Este requisito implica estudiar los distintos *frameworks* web existentes durante la fase de selección de tecnologías.

- **Asincronía.** El asistente de corrección debe poder procesar un gran número de entregas, especialmente en asignaturas con un gran número de alumnos matriculados. Por este motivo, el lenguaje de programación utilizado debe permitir la ejecución concurrente de distintos procesos. Muchos lenguajes incluyen mecanismos de programación asíncrona, que permiten realizar operaciones relativamente largas sin detener la ejecución del resto del programa.

Con estos requisitos en mente, se procedieron a analizar las fortalezas y debilidades de las tecnologías enumeradas a continuación:

- **Tecnologías basadas en la máquina virtual de Java (JVM)**. En este apartado se encuentran todos los lenguajes que generan código *bytecode* ejecutable en la máquina virtual de Java. La ventaja de este ecosistema es la interoperabilidad. Aunque el asistente estuviese implementado en Java, sería posible la incorporación de *plugins* escritos en otros lenguajes tales como *Scala* [OSV16], *Kotlin* [JI16] o *Clojure* [AGH+16]. El lenguaje Java fue la opción escogida para la herramienta verificadora de entregas [HR17]. Sin embargo, el hecho de ser un lenguaje fuertemente tipado tuvo un impacto negativo en la implementación del sistema de carga de *plugins*.

- **Tecnologías basadas en JavaScript**. El lenguaje JavaScript [Bro16] fue concebido inicialmente para su ejecución en un navegador web, pero en la actualidad es posible ejecutar programas JavaScript fuera del navegador mediante el entorno de ejecución Node.js [Mar18]. La amplia difusión de este lenguaje facilitaría la creación de *plugins* del asistente de corrección por parte de otros/as desarrolladores/as externos/as. Por otro lado, existe un framework (Electron [Kin18]) que permite el uso de tecnologías web para desarrollar aplicaciones de escritorio. Sin embargo, las aplicaciones generadas consumen demasiados recursos, tanto en memoria como en espacio de disco utilizado. Además, el modelo de programación utilizado en Electron está basado en comunicación interproceso. Esto dificultaría una posterior migración del asistente a una aplicación web.

- **Tecnologías basadas en Python.** Este grupo incluye, además del lenguaje Python [Lut13] en sí, los distintos *frameworks* web existentes para este lenguaje, tales como *Django* [KB18] o *Flask* [Gri14]. Las principales fortalezas de este lenguaje con respecto a los requisitos mencionados anteriormente son la posibilidad de cargar módulos dinámicamente, y su amplia difusión. Sin embargo, pese a que el lenguaje permite la ejecución asíncrona de procesos, su integración en los frameworks web aún está en temprano desarrollo.

- **Tecnologías basadas en la máquina virtual de Erlang (BEAM).** En esta familia podemos citar los lenguajes Erlang [Arm13] y Elixir [Tho18], y el *framework* Phoenix [MTV19]. Estos dos lenguajes de programación son funcionales, dinámicamente tipados, y orientados a la programación concurrente. Al igual que los anteriores, soportan de manera sencilla la incorporación dinámica de módulos. Además, el modelo de programación concurrente basado en actores permite la ejecución asíncrona de funciones. Por otro lado, el *framework* web Phoenix permite la creación de aplicaciones web en Elixir utilizando el mismo modelo concurrente que Erlang/Elixir. El principal inconveniente de estos lenguajes es su escasa difusión hasta el momento. No obstante, ambos poseen mecanismos de interoperabilidad con otros lenguajes (C, Java, Javascript, Python) utilizando el mismo mecanismo basado en procesos.

Aunque varias de estas tecnologías satisfacen, en mayor o menor medida, los cuatro requisitos expuestos anteriormente, hemos escogido el lenguaje Elixir y el *framework*

Phoenix para el desarrollo del asistente de corrección, principalmente debido a la experiencia previa en estas tecnologías por parte de algunos de los desarrolladores del equipo de este proyecto.

Modelos de visualización y componentes

Uno de los objetivos del asistente de corrección es la navegación entre los distintos componentes de una entrega. En esta primera versión, el concepto de *componente* de una entrega se refiere, fundamentalmente, a un fichero o directorio. Es decir, el/la usuario/a puede navegar entre los distintos directorios de una entrega. No obstante, en un futuro queremos permitir la navegación por otras entidades distintas. Por ejemplo, si la entrega de un ejercicio consiste en un fichero con código fuente escrito en Java, el asistente debería poder navegar por las clases, métodos y atributos definidos dentro de este fichero. En ambos casos, consideramos que estos elementos son, también, componentes de una entrega. En la herramienta de validación desarrollada anteriormente [HR17] se tuvo en cuenta este requisito desde el principio. Para ello se introdujo el concepto de *entidad*. Cuando un/a desarrollador/a implementa un *plugin* en el sistema, debe definir los siguientes elementos:

- **Tipos de entidad** introducidos por el *plugin*, si procede. Por ejemplo, el *plugin* de ficheros introduce el dos tipos de entidad: *fichero* y *directorio*.
- **Símbolos de función** para ser utilizados en las condiciones de una entrega. Por ejemplo, el *plugin* de ficheros introduce, entre otras, la función `files-rec`, que devuelve los ficheros contenidos en un directorio.
- **Símbolos de predicado**, también utilizados en las condiciones de una entrega. Son funciones que se evalúan a un valor booleano (cierto o falso).

El asistente de corrección involucra dos nuevos tipos de elementos, que han de ser introducidos por los *plugins* correspondientes.

- **Propiedades**. Una entidad dispone de varias propiedades visualizables por la herramienta de corrección. Por ejemplo, entre las propiedades de un fichero podemos citar su nombre y su tamaño. Para cada tipo de propiedad se ha de indicar, a su vez, un modelo de visualización que determina cómo se muestran los valores de esta propiedad en un documento HTML, y un valor booleano que indique si la propiedad es clave o no. Cuando dos elementos de una colección coinciden en sus propiedades clave, se muestran agregados en la vista de grupo (Figura 2). Por ejemplo, la única propiedad clave de un fichero es su nombre.
- **Acciones**. El asistente debe poder realizar distintas acciones sobre los elementos seleccionados. El conjunto de acciones soportadas depende del tipo de entidad al que pertenecen dichos elementos. Por ejemplo, el tipo de entidad directorio proporciona la acciones *Ejecutar comando de shell*, *Renombrar* y *Eliminar*. Un *plugin* debe especificar las acciones soportadas por cada tipo de entidad, y los parámetros que reciben cada una de ellas, en caso de necesitarlos.

6. Anexo

Referencias

[AGH+16] Jeremy Anderson, Michael Gaare, Justin Holguín, Nick Bailey, Timothy Pratley. *Professional Clojure (1st edition)*. Wrox. ISBN 978-1119267270 (2016)

[Arm13] Joe Armstrong. *Programming Erlang: Software for a Concurrent World (2nd edition)*. Pragmatic Programmers. ISBN 978-1937785536 (2013)

[Bro16] Ethan Brown. *Learning Javascript (3rd edition).* O'Reilly Media. ISBN 978-1491914915 (2016)

[Gri14] Miguel Grinberg. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media. ISBN 978-1449372620 (2014).

[HR17] Laura Hernando, Daniel Rossetto. *Asistente de corrección y validación de ejercicios.* Trabajo de fin de grado. Facultad de Informática. Universidad Complutense de Madrid (2017).
https://eprints.ucm.es/44426/

[JI16] Dmitry Jemerov, Svetlana Isakova. *Kotlin in Action*. Manning Publicacions. ISBN 978-1617293290 (2016)

[KB18] Jake Kronika, Aidas Bendoraitis. *Django 2 Web Development Cookbook.* Packt Publishing. ISBN 978-1788837682 (2018)

[Kin18] Steve Kinney. *Electron in Action.* Manning Publications. ISBN 978-1617294143 (2018)

[Lut13] Mark Lutz. *Learning Python.* O'Reilly Media. ISBN 978-1449355739 (2013)

[Mar18] Azat Mardan. *Practical Node.js: building real-world scalable web apps (2nd edition).* Apress. ISBN 978-1484230398 (2018)

[MSV+17] Manuel Montenegro, Clara Segura, Fernando Rosa, Santiago Saavedra, Laura Hernando, Daniel Rossetto. *Validación en línea de aspectos formales y corrección asistida de ejercicios en asignaturas con evaluación continua*. Proyecto Innova-Docencia 49/2016 (2017)
https://eprints.ucm.es/43643/

[MTV19] Chris McCord, Bruce Tate, Jose Valim. *Programming Phoenix 1.4*. The Pragmatic Programmers. ISBN 978-1680502268 (2019).

[OSV16] Martin Odersky, Lex Spoon, Bill Venners. *Programing in Scala (3rd edition)*. Artima Inc. ISBN 978-0981531687 (2016)

[Tho18] Dave Thomas. *Programming Elixir 1.6: Functional, Concurrent, Pragmatic, Fun*. The Pragmatic Programmers. ISBN 978-1680502992 (2018)