DevEval, porting e-learning technologies to lab examinations

Nuno Miguel Roque Gomes (MEEC - 65448)

Abstract—With the growing of e-learning technologies, more and more tools are being built in order to help assessing students’ performances. But one area which is not being properly focused concerns laboratory examinations of programming.

The existing tools that focus on programming development are more oriented towards long duration projects rather than smaller assessments, leading to a lack of assistance and support for lab examinations.

This work focuses on allowing the electronic assistance of lab examinations. DevEval was built including the core modules of a Learning Management System platform, but integrates those capabilities with Integrated Development Environments in order to facilitate the development of lab examinations. Comparing with existing tools, DevEval, which has an open client-server architecture, allows the storage of multiple answers per question.

Index Terms—E-learning, LMS, development assessment, development history, lab examinations

1 Introduction

With the increasing use of technologies and the everyday appearance of new ones, it can be considered important to use them to support and complement teaching and education.

Instructors and students alike can benefit from e-learning. It doesn’t have location restrictions and, compared to traditional learning, has much less time restrictions also.

It is even easier to have lectures or interactions with field specialists than having to bring them to a specific geographic place to talk in a class or a corporate seminar.[1]

When looking for e-learning solutions it can be impressive the amount and variety of existent tools, with each one either trying to specialize in a specific aspect or being as generic as possible. Still there are some blind spots.

Most of e-learning platforms existent nowadays allow the creation of the most generic possible forms of assessment. These tools give the most possible liberty to that process so instructors can elaborate their assessments how they want, with the complexity level they desire, etc. There is, at least, one type of assessment that, although can be forcibly adapted to fit these forms, when using existing platforms instructors aren’t allowed to have the best possible way to assess the students. This type of assessment is referred as lab examinations.[2]

Theses assessments are the ones characterized by a somewhat simple enunciation, with a certain number of consecutive objectives students should achieve in a short period of time, that is normally inferior to the duration of a class.

The generic assessment’s forms that existing platforms provide (like multiple choice, free text, linking words and/or images, etc.) aren’t appropriated for lab examinations. Adapting one of those would imply that students would forcefully use that platform’s application to answer them, and not an IDE (Integrated Development Environment), as usual, losing all the features these provide.

It was also observed that existent platforms would only store the latest answer for each question even though most of them allow students to repeat assessments, if instructors define so. This sounds reasonable for the usual assessment types but with lab examinations, having an history of what was done, could be beneficial for the formative evaluation.

This work provides a solution for helping instructors with lab examinations evaluation. It was built DevEval, a server-client architecture that allows instructors to create lab examinations and IDEs plugins to retrieve those assessments and show them to students, whilst also saving
their answers, even when it is sent more than one answer per question.

2 STATE OF THE ART

The e-learning subject has already more than 30 years of existence.

Although the term “e-learning” came to existence only in 1999, when it was utilized at a Computer Based Training (CBT) systems seminar, the concept has been documented throughout history.

When the first personal computers appeared, in the 1980’s, people had it easier to learn about particular subjects and develop certain skills and within the next decade, with the appearance of the internet, virtual learning tools began to thrive since several schools had set up the delivery courses online, reaching people with time and geographic constraints.

With this progression in the industry was necessary to define standards and specifications and one of the most important was the IMS QTI[3].

QTI, that stands for Question and Test Interoperability, is a specification that is the standard in the industry nowadays. It is a specification that enables the exchange of data heterogeneously between all systems and tools compliant. The binding of the QTI abstract models is made with XML, eXtensible Markup Language[4]. For each question element, named as item, QTI allows the attachment of a set of rules used to process the responses to that item. It allows also, on the assessment level, the definition of score rules and of score aggregation.

It is important to introduce QTI because of its role in the industry as the major specification for e-learning items.

Several studies have been made relating programming with human cognition[5]. Cognitive processes include perception, attention, memory usage, understanding, problem solving and reasoning. These operations are also performed by a CPU with associated memory in order to obtain an output and like in a computer, information goes through different stages of cognitive processing and storage to give an output. It is also represented in symbolic form, as in computers.

There is an analogy between compiled programs and automatic processes (cognitive-wise) since both are carried out directly[6]. On the other hand, controlled processes are similar to interpreted program code that still requires translation into machine terms at runtime. These require attention and are of limited capacity.

In a study made in University of Kent[7], regarding first-year C.S. students, was recorded and analyzed their programming development. Recurring to BlueJ[8], a pedagogic IDE intended to support the learning of Java, whenever students compiled their programs, "snapshots" of those were taken: it includes the complete source code, output information from the compiler, metadata, etc.

Students get the assignment, next week they may have to deliver their solution and the next they receive their grade. The problem is that instructors only have a single snapshot, the one representing the final product; they can’t infer from that if students completed the assignment quickly or if they struggled for some time.

Relating this reasoning to the cognitive process in programming, it is our understanding that allowing the storage and observation of multiple answers per question, or in other words multiple “snapshots” of students’ code, it is an asset and not a flamboyance. Even though some of the instructors that would use DevEval might not use that option, it is still present for those who do.

2.1 Existing Tools

In this section are presented existing tools and platforms that could pose a possible solution, or at least a partial one, that could be used, or are similar, to the implementation of DevEval.

2.1.1 Version Control Systems

Version control systems, VCS, execute correctly a solution to one of the presented problems: managing multiple versions of the same file. In terms of applicability in e-learning that would mean having multiple answers per question.

When using these tools, programmers commit their code to a repository whenever they feel it is appropriate (e.g. correcting a bug,
adding a new feature, etc.)[9]. They can also synchronize their commits with their colleagues. This is one of the main benefits of using VCS: when working in a team, programmers won’t be stepping on each other’s code, since the system will either merge the changes, between commits made by different people to the same files, or warn every actor about it. Every time a programmer commits a change it is created a new version of the corresponding file, hence the multiple versions per file, the feature referred previously.

In summary there are a lot of VCS out there and each one has its own pros and cons but, although they can manage correctly multiple versions of the same file they do not answer to the global problem presented, they only answer to part of it, even though the majority have IDE integration. The problem with these tools is that they are not LMS and so they don’t have the e-learning common architecture, neither they apply e-learning specifications.

From these tools what can be important to withdraw is the technology and methodology of managing multiple versions of files.

2.1.2 E-learning

E-learning offers the ability to share learning material, to conduct online classes and to communicate with professors via chat or others message applications[10]. Most of the times it is a free solution to provide users the ability to learn in a personal tempo and around each ones lifestyle. It can be used by someone that has all day available to learn new subjects and topics, or by someone that only has one hour a day. Overall, and comparing to traditional learning, e-learning can be cheaper, less time consuming, or at least with a broader time span and potentially better.

2.1.3 LMS

In this section will be evaluated some existing LMS tools that could provide answers to the proposed problem.

The general objective, that all these platforms have, is to provide a featured tool that implements e-learning technologies and specifications and that is simple and intuitive for the end-user. It can be seen in figure 2.1.3 the standard model for the architecture[4] of LMS platforms.

The core components, that every LMS tool should have are:

- **Database** where all the assessments, tests, questions, students’ answers, etc. are stored;
- **Assessment Manager** that manages all the queries for the mentioned items in the previous point;
- **Delivery System** that manages how students’ answers are dealt with;

Fig. 1. Generic architecture of a LMS platform

The other systems are important but not crucial, although it can be fundamental to have at least either the **Authoring** system or the **Import/Export** one, in order to create or import items to the database. Most of the LMS tools nowadays feature all the illustrated systems by default, specially the Authoring and Grading ones.

Within the LMS class of systems, one of the most widely known is probably **Moodle**[11]. This engine enables educators to create dynamic courses. It has a modern and easy to use interface, both for desktop and mobile.

Although **Moodle** provides an **API**[12] for development, it is mainly for developers to create plugins and web-applications to complement the core engine, as it can be seen by quickly looking through the list of core **APIs**. Although
its API has some REST protocols, it is not a Restful one.

It can be concluded that, although Moodle is one of the most developed, important and feature-rich LMS engines out there, it doesn't really solve the presented problem, as it focuses solemnly on web-based applications, thus disabling the IDE integration. Also, the engine cannot store multiple answers for the same question. Although it is possible to answer multiple times to an assessment, only the last answer for each question is stored.

Another LMS engine studied was TAO[13]. During the process of finding a tool to solve the proposed problem, TAO appeared to be an almost complete solution since it had an API with some of its calls being REST. TAO is an open source engine that started to be developed in 2002 as a joint project between the University of Luxembourg and the research center Henri Tudor. Although it has been developed for more than 10 years, TAO was still in a beta state (when development began), specially its API.

Since this engine had an API with some REST calls, it could be integrated in an IDE easier that most other LMS engines. When evaluated, it seemed that the only system that TAO didn't have was the multiple answers storage (for the same question), and so it was decided that it was the best option of existing engines, to use as a base where the unfulfilled characteristics would be developed upon.

All evaluated platforms provide the LMS' core modules and, although all of them have a multitude of formats for the creation of assessments, none has a specific format for lab examinations. Is interesting to refer that some tools already facilitate external integration, but as it was discovered, the delivery system isn't usually between the APIs available. To finalize, as it was also expected, none of them deals with multiple versions of the same file, or in other words: multiple answers per question.

2.1.4 Automated Programming Assessing systems

There is another field of e-learning, more focused on programming development, that tries to provide automated programming evaluation. These systems don't have a base architecture similar to LMS tools but they are mostly adapted to be integrated with IDEs and to evaluate and analyze code. It will only be analyzed one, the one that was considered most relevant.

Based on the premise that one of the problems with Computer Science (CS) courses is that instructors only had the final submission of students assessments to evaluate their knowledge and skills, it was built Marmoset[14], an Eclipse plugin to address the lack of feedback between students and instructors. It had two main features:

- Use a CVS repository to store the complete state of students' projects every time they saved, added or removed files, producing a fine-grained history of the development of each project.
- The plugin allowed students to easily submit their projects to a central server that would perform tests on each submission and return feedback on how that submission performed in face of the requirements.

Many other automated assessment systems exist and all perform similar tasks: they receive a set of rules and test cases (from instructors), projects (from students) and assess those projects according to the defined rules and test cases. Marmoset was one of the few that could also store and display the development history. Like Marmoset, and working on a similar faction, exists ClockIt[15] and Syde/Replay[16], [17]. Both systems have the same problems as Marmoset. Although they all register and store the history of the students' development and can be integrated with IDEs, none of them has the main capabilities of a LMS platform: they don't provide an Authoring tool neither the Delivery system.

2.2 Evaluation

With this analysis it was understood that Version Control systems are effective in managing multiple files and are easily integrated with other tools, plugins, etc., but they are not e-learning tools directly.

LMS tools are more close to what is pretended since they have a database and systems for managing and delivering all the assess-
ments, which are created in the Authoring systems by the instructors, and some of those tools even have an API that facilitates development of applications to interact with the existing system. Although some of them allow external integration, it was concluded that those APIs don’t allow alterations to the Delivery system in order to implement multiple answer per question, as pretended for the solution. Besides this, none of the evaluated LMS tools as assessments forms for lab examinations.

The last class of systems analyzed was the Automated Programming Assessing tools. These are solely focused on assessing programming development, which solves one part of the presented problem. The most part of tools in this class, even have some degree of external integration since they are applied directly onto an IDE in order to obtain students code. It is understandable that those who do, also allow multiple files.

In the end none tool of those evaluated and none of the existing, as far as our knowledge goes, can implement all the requirements needed to achieve a solution for the proposed problem.

2.2.1 TAO

As mentioned, TAO’s engine API was in a beta state and that posed a lot of challenges. The documentation was also incomplete and simplified. This forced the study of the engine through it’s code, in order to understand how the API worked, and a great amount of trial and error testing in order to integrate the engine with an IDE.

Although TAO provides a plugin system, the integration of a new one with its default systems and tools isn’t linear or at least not correctly documented, which led to failure when trying to implement a plugin that would interact and change attributes from different default systems of the engine.

The main objective was to obtain REST calls from the Delivery System to be used by an IDE. Unfortunately this part of the engine was one of the few that hadn’t, yet, the REST calls on it’s API. Because of these two issues, it was necessary to change code, from default systems, to obtain what was intended.

The standard API for each system was completely focused on web-applications, like the other studied engines, and since the Delivery system hadn’t a REST API, it was necessary, when editing the code mentioned above, to extract information from html code obtained through REST calls to other systems of the engine.

Besides all these challenges presented, that were appearing along the process of creating the integration with an IDE, there was also the challenge of implementing a script that would “install” the alterations to the default code, that affected multiple systems of the engine, and the challenge to create a system or plugin that would allow to store more than one answer to the same question.

In the end, all the challenges led to abandon TAO as part of the solution to the presented problem. Even though some were already surpassed, most of the solutions found were hard-code ones, like what was referred previously (having to extract information from html code).

3 DEV_EVAL

3.1 Objectives

It is presented next, in a visual and simple manner, the objectives this work proposes to achieve:

- Be oriented towards lab examinations (not projects);
- Create a simple LMS tool that:
  - Has all the core modules of a LMS platform:
    * Database;
    * Assessment Manager;
    * Authoring;
    * Delivery:
      ⋅ Deliver assessments to authorized applications;
      ⋅ Deliver students’ solutions to instructors;
  - Allows the integration with external applications (through REST calls);
  - Is capable of storing multiple answers per question, per student;
- Create an IDE plugin that demonstrates the potentialities of the developed LMS tool;
The tool that was built to fulfill these objectives was named *DevEval*.

### 3.2 Architecture

In figure 3.2 can be seen an illustration of the architecture designed.

![DevEval's architecture illustration](image)

There is a central module, the Delivery system, that is a pivot that mediates communication among all other modules. It obtains information from the LMS Authoring and Manager and stores it in the Assessments Repository. When an application requests an assessment the Delivery System fetches it from the Repository and delivers it.

The complete system is presented in the form of a server with each of the referred modules. The only one with which client applications interact is the Delivery System itself and it is this one that communicates with the other two modules when necessary.

### 3.3 Data Model

Generally DevEval’s data model is similar to every other LMS. The big difference is that DevEval’s courses, students and assessments are integrated with an Academic Management Software (*Fenix*) and are not just part of the tool. Figure 3.3 illustrates the data model.

![Representation of the data model](image)

### 3.4 Implementation

Our LMS tool, called *DevEval*, is composed by an **Authoring System**, that allows instructors to create assessments via their web-interface, an **Assessment Manager** that manages the stored assessments, questions and answers, a **Delivery System** that allows the request of the assessments and receives the answers, and a **Database** where everything is stored.

The engine has two components:

- Server
- Client for Instructors
- Client for Students

The server is composed by the systems listed above while the client has two different interfaces: one for the learners and another for the instructors.

Overall the server was built using *Spark*, a framework that allows the creation of web application using Java 8[18]. For the applications examples, that would provide a visual demonstration of the server capabilities, it was created a plugin for *Eclipse* (an IDE) and the server was also prepared to receive HTTP requests and so any browser can be an application that communicates with the DevEval’s server. The *Eclipse* plugin is the interface for students while the web-interface is for instructors.

In terms of requests, the server answers to HTTP (and those related to the designed web-pages, like images, javascripts, css, etc.) and JSON ones.

It will be described next, how each of this components was implemented and what characteristics they have. After that it will be discussed the communication channels used, the
integration with Fenix API (Fenix is the Instituto Superior Técnico, IST, academic and administrative management software) and a final evaluation of the engine built.

3.4.1 Server
Since the core part of the data used are files (students’ code, assessments, questions, answers, etc.) the database was created based on directories. This means that each course has it’s own directory, inside that directory are listed all semesters where the engine was used and inside those directories are listed the directories pertaining to each student enrolled and a directory with all the assessments created in that semester for that course. Inside each students’ folder it is listed the directories pertaining all assessments they did. To finalize, inside the last folder, in the hierarchy, are all the files that contain the answers obtained from the IDE plugin during that student resolution of the specified assessment.

In the end, directory-wise there are two path to be followed:

- Course/ Semester/ Assignments/ Assessment
- Course/ Semester/ Student/ Assessment/ Answer

The way DevEval’s database was built is similar to a MongoDB[19] one, which stores data in the form of documents. The core fundamentals are the same.

The server addressed http connections and was implemented to be Restful (in section 3.4.4 this is better described). The server’s answers either are in HTML or JSON, specifically the first one when dealing with the web-application client (the one for the instructor) and JSON for the IDE plugin (the interface for students).

The Assessment Manager deals with all the requests related with acquiring assessments, questions and answers, either from the web-application or the plugin. Each one will tell the server what resource the user wants. The plugin can request assessments and its questions while the web-application can access all three types of resources. As mentioned in section 3.3, all the files stored are JSON formatted, so every answer to requests of these resources would send a JSON answer that will concatenate a boolean “success” parameter with the information necessary from the file requested (the title and number of questions if was requested a specific assessment, the question id and the question itself if it was requested the question, etc.). If, by one of multiple reasons (a problem connecting to Fenix, failing to read the requested file, etc.) the server is unable to respond accordingly, it will send a message that has the “success” parameter as false and adds a string “reason” parameter, explaining why the request failed to be answered as expected.

When the plugin requests a list with available assessments, the server will start by verifying in which courses the student is enrolled in order to know if he is authorized to perform that assessment, resulting in a list off all the assessments from courses the student is enrolled. After that the server will verify which assessments, from the abbreviated list, have the start and finish date framed with the current date. The resulting list is the one sent back to the plugin.

The Delivery System deals with all the information that the plugin sends to the server (http POST requests). For every of these requests received the server will verify if the student has already started the defined assessment and if it hasn’t, it will create a new folder for that assessment in the student one (in the hierarchy presented previously in this section).

The default name for the files sent is a defined string plus the question number plus the submission counter value (e.g. “submission-1-0”, would be the first submission related to the second question, with id one; question ids and the counter start with zero). With that in mind, after the mentioned verification, and since the server knows the question that the message sent is related to, it will verify if any answer for that question already exists. If it doesn’t, it will create the first file, with submission count equal to zero. If it does, the server will verify how many submissions exist while it increments the submission counter. Then it stores the new answer, that already comes JSON formatted and ready to be stored with the appropriate name in the
appropriate path (e.g. "Database/CourseA/14-15-Sem1/Student12345/Test1/submission2-1.json", this would be the file to be stored from the second answer to the third question of the assessment "Test1", by the student with id "12345", enrolled in the course "CourseA" in the first semester of 2014/2015).

Related to the Authoring System, it is presented to instructors in the form of a web-interface, programmed with HTML, CSS and JavaScript, that allows them to create assessments. Whenever one is created, the server will then store it in the correct path.

3.4.2 Instructor Client

As mentioned the client part of this work is divided in two: one interface for students and another for instructors. In this section will be discussed and explained how both work.

The instructor interface is a web-application. Although the server has resource identification through URIs, the simpler way to use is just interacting with the webpage. All functions that are available through URI are also implemented through the webpages.

When the instructor presses the "end" button, a request is sent to the server with all the data obtained already processed to JSON by JavaScript code.

The instructor starts with accessing the domain of the system. There he has the authentication screen, with a introduction to the project. After authentication is validated, the instructor has a search-box where he can start writing the course name, and, in real-time, will be listed courses with the letters already written in the search-box. In front of every course appear two buttons: one to access the data of that course and the other to create a new assessment.

When he presses the later, a new webpage appears: it will have at least four textfields: one for the input of the title of the assessment, another one for the first question (since it is mandatory at least that one is created), and the last two for the starting and finishing date for the assessment to be available. All these four fields are mandatory. Besides that, are present two buttons: one that ends the process (verifying first if all the mandatory fields are correctly fulfilled) and another one to add a new textfield in order to add more questions to the assessment.

In the case he chooses the access button, he will be presented with a list of all the semesters the chosen course has already had assessments created. This gives instructors flexibility to learn how previous assessments did: which questions were assigned, how students responded, etc. Whichever semester he chooses, next will appear a list of all the enrolled students and an option to access the assessments defined for that course in the chosen semester. When chosen the assessments option, will then appear a list of all the assessments already assigned and pressing one of those will lead to a list of the students that already answered the assessment selected. Choosing one of them will lead to the presentation page, that will be discussed further down.

When the instructor selects a student, after the "choosing semester" menu, will lead to a list of all the assignments that student has already answered. Selecting one will lead to the presentation page. What happens with the two options discussed right above is that instructors can go either way: they can go through a specific assignment and see which students have already solved it, or, by choosing a student, see which assignments, from the selected course and semester, has he already solve.

The mentioned presentation page, is a page where instructors can see all the answers submitted throughout the selected student resolution process. The page has two main areas. The first presents a question and there are two buttons - "previous" and "next" that will gather the information for another question. The second has a presentation similar to the first, but instead of showing the question, it shows the answer pertaining the question in the area above. The "previous" and "next" buttons contact the server to retrieve the correspondent answer, if it exists.

To note that all the changes in the website are made through different URIs, except the first page where instructors can search for the courses.
3.4.3 Student Client
The interface for learners is a plugin for Eclipse. Since the focus of this tool is to integrate e-learning with lab examinations, that seemed the correct way to do it. When the learner opens the IDE, it will be present in the toolbox the DevEval plugin. When clicked, it will open a pop-up window that will prompt for authentication. Since this is a project for a Master Thesis in Instituto Superior Técnico, and as referred, we recurred to Fenix API, was only natural that this example plugin would abide by IST’s id system, thus students had to authenticate with their student id.

When authentication is successful, it will be presented a list of all assignments that the learner can realize.

After choosing one, will be presented the first question from the selected assessment. There are also two buttons: “previous” and “next” that will request the next or previous question, when available. When the learner presses “next” and it is the last question, a warning will pop-up giving him the choice to go back and change his answers, or finishing the assessment.

As mentioned before, the plugin will send the answers to the server. This happens every time the learner pushes one of the two buttons. Automatically the plugin will gather all code from all open files from all open windows from the defined workspace, group that information in a JSON file and send it to the server, asking next for the pretended question. What this means is: when a student goes back to change something from a previous answer, there will be more code than the necessary to solve that question, but that gives instructors the information that the learner only did the alteration after seeing/solving the next question.

In the end the last answer to the last question will be the final submission.

Note that during all these processes the window where all the referred information appears is always the same. Its content is always being updated regarding the requests made.

3.4.4 External API
The communication channel used between server and client was http. GET and POST requests where used for various actions. The server was also implemented as a Restful web service:

- **Resource identification through URI**: the URLs would identify the resources need (e.g.: “/course/semester” will list the students enrolled to the defined course in the defined semester);
- **Uniform interface**: resources are manipulated using a fixed set of operations, with the use of POST and GET requests.
- **Stateful interactions**: every interaction with a resource is stateless, that means that request messages are self-contained, they have all the information needed so the server can answer correctly and the server never stores state values.

One other field of a Restful web service is the self-descriptive messages, where “resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others”[20], but this was not completely applied. Our engine could respond to some formats but the resources dissociation was not completely present, the client would receive either a JSON answer or a HTML one.

3.4.5 Fenix Integration
The LMS Authoring and Manager systems are part of a built server that interacts directly with Fenix. Fenix is the Instituto Superior Técnico, IST, academic and administrative management software. Since this dissertation is developed for a masters degree from IST it is only logical to integrate part of this platform in this work, specially because it has an API that provides information about courses, personal, etc.

3.5 Evaluation
Looking to the problem presented and that this work focused on solving, it can be said that the core features were implemented successfully. DevEval is a simplified LMS engine, focused towards lab examinations, but that didn’t disregard the core functions of these type of systems.

The main focus was on its Delivery System, since it had to be built to allow IDE integration.
The achieving of this objective is exemplified with the plugin developed for Eclipse that can, as pretended, contact the server in order to obtain assessments and its questions and also to send the answers. It is also possible, through the built API for other plugins or extensions (for other IDEs) to be built around DevEval. The other main focus of this work was that the Delivery System allows multiple answers for the same question to be stored, which was another objective achieved.

The fact that the API provided by DevEval allows everyone to build an IDE plugin, influenced the decision of using JSON for the assessments and answers, instead of QTI. If the later was chosen, those who would want to build a plugin to interact with DevEval, would have to make it QTI compliant, which ain’t simple or linear: it as a big set of rules that need to be fulfilled, which could demote some developers.

4 Conclusion

It was found a “blind spot” in existing LMS platforms and e-learning tools, since there is none that focus the assessment of lab examinations.

Although it could be debated that learners can just write their code in textfields in a standard LMS engine, the truth is that, for teaching purposes it is important that they have basic auxiliary tools, like those provided by IDEs. For these reasons we thought it was essential for a programming oriented LMS to be capable of integration with one.

For this reason it was considered that existing tools weren’t able to perform what was needed and so it became the solution to create a new LMS tool that could prove the usefulness of those characteristics when assessing lab examinations.

The created tool, DevEval, is capable of integration with IDE in order to send assessments’ data and to receive answers, which could be for one or multiple questions, since the system allows more than one answer per question. To finalize the process, the tool allows instructors to visualize students’ code history through the web-interface.

References