Promoting coding best-practices by extending Moodle's VPL

Marílio Cardoso
ISEP - Polytechnic of Porto
(Portugal)
joc@isep.ipp.pt

Rui Marques
ISEP - Polytechnic of Porto
(Portugal)
rfm@isep.ipp.pt

António Vieira de Castro
ISEP - Polytechnic of Porto
(Portugal)
avc@isep.ipp.pt

Abstract - **This paper translates an effort of improving VPL functionalities in several aspects and fields, that may be considered conceptual. The context is APROG (unit course of Algorithm and Programming), at School of Engineering (ISEP), Polytechnic Institute of Porto (P.PORTO), that uses VPL to automate, normalize and simplify student's tests of a specific set of Algorithms. These tests are factual (built in on VPL), which tests if results meet requirements. Extended VPL tests validate format, content, and specific exercise check-points. The overall objective is to help the student achieve the overall solutions that includes class oriented, concept followed by student deployment, with VPL validation of results, but also, design. In simple terms, not only if the exercise is solved, but also if it is solved following the class resolution guidelines.**

Keywords – Coding Standards, Java Best Practices, Software Development, VPL.

INTRODUCTION

VPL stands for Virtual Programming Lab [1], it's responsible for managing programming assignments in Moodle (a popular support tool for teachers and students in academic environment). It operates as a module and it's highly integrated allowing: students to create, edit and run source code directly on their browser window; teachers and students to debug and run interactively their code; teachers can review students results; some types of plagiarism control (disabling copy-paste and validating similarity). An evaluate procedure can be ran to deliver a percentual evaluation. Evaluation can be incremental, decremental or even limited to a maximum amount of tests that can occur [2].

VPL is a rather robust technology, with a group of tools (or subsystems) that allow arbitrary code to run, without any server administration intervention, in a safe and isolated manner - relying in a Jail Server - this way user test their algorithms, while other system functions, communications and user data is not accessed. This is achieved by a set of temporary generated features that Jail Server uses in order to maintain VPL user data, without osmosis with the actual system, while using it's binary executables, it's library class files and remaining files that may be required by the selected compiler.

In APROG's case a Java compiler is used [3]. This operation needs a compiler and a runtime binary ("javac" and "java" respectively).

The Moodle module encapsulates the compiler program through a text string, that is converted into a script and then executed in run-time (lines 3 through 5). These lines contain the mentioned compiler and runtime library.

This type of "rendering" allows code sanitation but implements some limitations, each exercise must have it's own set of configuration and test cases files. In the test case example, this is a desired feature, in the configurations point of view, where design tests are assembled, this is a tremendous inefficiency. Each exercise has the same configuration framework to validate concept and design. However, if a change is made, improvement or bug correction, this correction, however simple, must be implemented in every exercise. Which leads to a "not-so-framework", were exercise #1 has implemented design checks A and B, while #2 has A.1 and C, and so forth, making version control a type of nightmare.

This process has now evolved to a centralized framework of design concept checks, that can be implemented horizontally in the VPL platform, specific configuration done one file at a time (as intended), while updates, upgrades can be done in a centralized manner.

Another VPL limitation is the inability to render visual Graphic User Interface libraries (Java Swing or Java FX for instances), disabling such features, and redirecting input/output through console window. This limitation is treated as a code re-write request, into making the code work with console I/O - Readers/Writers (Java Scanner and System Out).

This paper describes the evolution in this process and it's state of art.

METHODOLOGY

VPL is divided into four main files for each exercise: vpl_run.sh, vpl_debug.sh, vpl_evaluate.sh and vpl_evaluate.cases. In this scenario the attention is brought over vpl_run.sh and vpl_evaluate.cases.

The vpl_evaluate.cases is a simple file that may be complex to manage, which regards the way the program is processed, input stream and output sequence [4].

The output is validated in a form and function type of validation (Carbon-Copy Validation), requiring that the student supplies it's result as-is. This is a negative point for this technology, as the algorithm may be correct, but if presented in a different way from the one provided, a negative result is yield.

In the vpl_evaluation.cases it's possible to add Regular Expressions (RegEx) for the results window. This plasticize the process of validating results, however it is not a perfect fit. Considering the following scenario where the user submits the amount of numbers to be entered, and them prompts for that exact amount, ordering the array (descendant), and then rotating it to the right:

```
output =
How many numbers?
Number 1:
Number 2:
Number 3:
4 3 2
2 4 3
output = /[\r\n][\w ?]*[\r\n]([A-z 1-3:]*[\r\n]){3}([0-9 ]{5}[\r\n]){1}([0-9 ]{5})/
```

The RegEx validates effectively the position of the characters, number of occurrences and the places where specific characters should not occur. It also validates that the maximum number of characters to be outputted. This solution suits VPL needs in order to allow a plastic input messages and methods, but on the other hand, it does represent additional problems in managing exact results, as it does not integrate with supplied arguments (i.e. amount of "Number"'s' to be inserted) and also the expression must be designed with explicit results:

```
output = /[\r\n][\w ?]*[\r\n]([A-z 1-3:]*[\r\n]){3}(4 3 2[\r\n]){1}(2 4 3{1})/
```

Which can be roughly translated into applying specific result set and input text, as VPL does not allow a hybrid approach Regular Expression + Carbon-Copy Validation.

In such a context, our strategy reverted into Carbon-Copy Validation followed by exercise focused resolution technique control. This applies to the following control points:

- Java Class name and filename must match;
- Alerting for the forbidden use of Java Swing or Java FX;
- Search for a specific function name;
- Account total number of lines where the program is expected to run using less lines;
- Number of functions created for this particular task;
- Number of Constants;
- Number of "return"'s in a given function name;
- Number of lines of a specific function name;

The approach is to replace vpl_run.sh (in Fig. 1) with a particular set of commands that could scrutinize student code prior to runtime. This way, customizations could be applied, within Moodle/VPL program flow.

In the actual solution, we had to centralize code development, opting for a GIT format of deployment, unifying the program. Actual developments entitled "VPL Extensibility App" are in it's 1.0.2 version, and require a rather simpler model in "vpl_run.sh", composed by following sections: static declaration section, version management and file download section, parameters override section and, finally, validation, compile and run section.

This new strategy simplifies overall management of validation code, promoting adding new features while maintaining earlier exercises working. Also permits painless debugging of new versions by creating a local "vpl.version" that points to the new/unstable version.

Conclusions

The core operation of VPL is already a good starting point, merging teachers and students in one framework, allowing them to share, test and fine tune APROG exercises. This is already a very good thing. What we have tried, is to improve that functionality into an extended array of tools that can provide pre-compile, runtime tests, that suite each exercise specification. Improving class integration with VPL, better evaluation of APROG subjects with greater precision, and overall improvement powered by the continuous feedback provided.

References

[1] J. Rodríguez-del-Pino, E. Rubio-Royo and Z. Hernández-Figueroa, "A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features," International Conference on e-Learning, e-Business, Enterprise Information Systems, & e-Government, 2012.

[2] D. Thiébaut, "Automatic evaluation of computer programs using Moodle's virtual programming lab (VPL) plug-in.", Journal of Computing Sciences in Colleges 30.6, pp. 145-151, 2015.

[3] M. Cardoso, A. Vieira de Castro, Á. Rocha, "Integration of Virtual Programming Lab in a process of teaching programming EduScrum based", 13th Iberian Conference on Information Systems and Technologies (CISTI), Cáceres, Spain, 2018.

[4] A. V. Wanhenheim, J. E. Martina, R. L. Cancian and J. C. Dovichi, Developing Programming Courses with Moodle and VPL - The Teacher's Guide to the Virtual Programming Lab, Bookess, 2015.