Parameterized and automated assessment on an introductory programming course

Francisco de Assis Zampirolli[*], Paulo Henrique Pisani, João Marcelo Borovina Josko, Guiou Kobayashi, Francisco J. Fraga, Denise Hideko Goya, Heitor Rodrigues Savegnago

[1]Federal University of ABC (UFABC)
Av. dos Estados, 5001 – Santo André – 09210-580 – SP – Brazil

{fzampirolli, paulo.pisani, marcelo.josko}@ufabc.edu.br,

{guiou.kobayashi, francisco.fraga, denise.goya}@ufabc.edu.br,

heitor.rodrigues@aluno.ufabc.edu.br

Abstract. *The generation of individualized exams can contribute to a more reliable assessment of the students. Manually performing this procedure may not be feasible, even more on a large scale. An alternative to deal with it is the automatic generation of questions. This paper discusses an innovative solution to simplify test generation and correction through parameterized questions in the context of a four-month Introduction to Programming course under a blended-learning (IP–BL) approach. It combines the open-source tool MCTest with Moodle and VPL plugin to generate and also automatically evaluate parameterized programming language questions. We applied an intervention based on this solution in two IP–BL groups (a total of 171 enrolled students) using Java.*

1. Introduction

In programming course tests, Automated Assessment (AA) of questions is a topic that has been extensively discussed. By using AA, the student can receive feedback automatically. Empirical evidence justifies its use, including for Programming Exercises (PE), as the following paragraphs illustrate. Some key aspects are the improved motivation due to automatic feedback and higher grades obtained by the students.

Gordillo (2019) reported that AA improved the motivation of the students and their programming skills when compared to a group that did not adopt this approach. Furthermore, AA can lead to higher grades, as observed by Wang et al. (2011). They evaluated a system named AutoLEP in 4 classes comprising 30 students. The final result was an average grade of 79.3 for the students who used AutoLEP, against 73.2 when it was not.

Current paradigms for assessing students' competencies and skills in programming require tools capable of continuous assessment learning (Galan et al., 2019). In this sense, the authors presented an extensive study in a practical programming course from 2011 to 2018, in the Bachelor of Computer Science course, at the National University of Distance Education. They reported positive results from the adoption of AA.

[*]Grant #2018/23561–1, São Paulo Research Foundation (FAPESP).

Based on empirical evidence that supports the use of AA, this paper presents an innovative solution to simplify the generation and correction of *parameterized questions* (paper or online tests) that can be used by various educational institutions. A *parameterized question* contains code which is computed to generate several versions of the question, a concept related to *calculated questions*[1].

Although there are several tools for AA (Staubitz et al., 2017; Demir et al., 2010) and some online judges (e.g. BOCA[2] (de Campos, 2004), URI[3] (Bez et al., 1308)), to the best our knowledge, the number of studies dealing with automatic generation of individualized/parameterized questions is not that large. The main contribution of this paper refers to a novel proposal to integrate an open-source system named MCTest [4] (Zampirolli et al., 2019) and the Virtual Programming Lab (VPL) Moodle plug-in (Rodríguez-del Pino et al., 2012).

The remainder of this paper is organized as follows: Section 2 introduces some related work; Section 3 describes the methodology and our proposal for automated parameterized assessment; Section 4 shows and discusses its preliminary results; and, finally, Section 5 presents our main conclusions and opportunities for future work.

2. Related work

As discussed in the introduction, automatic feedback can increase the motivation to study. Gordillo (2019) presented a work carried out to verify the impact on the use of Automated Assessment (AA) tools in a web development course, in the third year of the Bachelor of Telecommunications Engineering using Moodle. They used a tool called IAPAGS (Instructor-centered Automated Programming Assignments Grading System) for automated correction. It was reported that AA improved the motivation of the students and their programming skills when compared to the group that did not adopt this approach. Another study (Venero and Mena-Chalco, 2019), which used Moodle and VPL for AA in a programming course, showed positive results.

Moreover, AA can lead to higher grades. In Wang et al. (2011), an automated learning and evaluation system named AutoLEP was presented. They evaluated it in 4 classes comprising 30 students. The final result was an average grade of 79.3 for the students who used AutoLEP, against 73.2 when it was not. Alemán (2010) also presented empirical results of using another AA system, named Mooshak, in a programming course. They concluded that the tool promoted student interest and produced a statistically significant difference in scores between the experimental and control groups. Later, Rubio-Sánchez et al. (2014) presented additional empirical assessments of the Mooshak system in a course of algorithms design and analysis.

Some studies also dealt with the generation of individualized/parameterized questions. The work from Radošević et al. (2010) proposed a method for creating individual questions using Programming Exercises (PE), in which the teacher must program

[1] *Calculated questions* "*offer a way to create individual numerical questions by the use of wildcards*", by `docs.moodle.org/en/Calculated_question_type`. At MCTest, wildcards can be texts, numbers, equations and figures.

[2] `ime.usp.br/~cassio/boca`

[3] `urionlinejudge.com.br`

[4] `github.com/fzampirolli/mctest`

his questions in C++. Students submit the codes in the Learning Management System (LMS) through the web interface, but without automatic correction. Another proposal is from DuFrene (2020), which proposed a method to create PE and applied it to generate exercises for the "loops" topic. Later, Hagiya et al. (2019) proposed methods to automatically generate programming questions, though it may require human intervention at some stages.

3. Materials and methods

A key aspect of our investigation is the proposal of a method to implement parameterized Automated Assessment (AA). It involves designing parameterized questions supporting AA and feedback. This section describes our attempt in this direction.

3.1. Dissertation questions

MCTest (Zampirolli et al., 2019) supports multiple-choice and dissertation (or essay) questions, in which the latter is the basis to describe Programming Exercises (PE). In the next section, two types of dissertation questions are described: *assessment with manual correction* (Section 3.1.1) and *assessment with code submission on the computer* (Section 3.1.2).

3.1.1. Dissertation questions: assessment with manual correction

As an example of a dissertation question, consider the topic "looping and table test" (the following question was applied in IP–BL course, Exam1, in 2019.1, in Section 4 this course is contextualized). A screenshot of the MCTest system and an English version are shown in Figures 1 and 2, respectively.

Figure 3 provides the complete description of the Table Test question, shown in Figures 1 and 2, including LaTeX text and Python code, as detailed below (the red text is the parameterized part of the question):

Part I. Description of question: Provide the description that will appear on the student exam, see Figures 1 and 2. Note that the description shows the text `a = [[code:a0]]` and `b = [[code:a1]]`. These values of `a = 16` and `b = 19` that appear in Figure 2 are random values, defined in Python, as detailed in Figure 3, see lines 2 and 3 of the `myRandom` function (Part IV.a). Variables `a0` and `a1` receive random values between 11 and 19 (python `randrange` function);

Part II. Table shown in Figure 2: As defined by variables `a0` and `a1`, it shows in the program description (left column in Figure 2, program to be simulated - left column in Figure 3), variations `a2` and `a3` (row 1 from the program in Figure 3, Part II.a), `a4` (row 7) and `a5` (row 13);

Part III. The correct answer on the back of the sheet: This part of the question description presents the template (correct answer) in the `mySimulation` variable. The content of this variable is defined in the algorithm function return, described below;

Part IV. Python code between "`[[def:`" and "`]]`": This part contains Python code to provide the contents of the variables that appear in the question description, as presented in the previous parts. Important: This Python code can be simulated

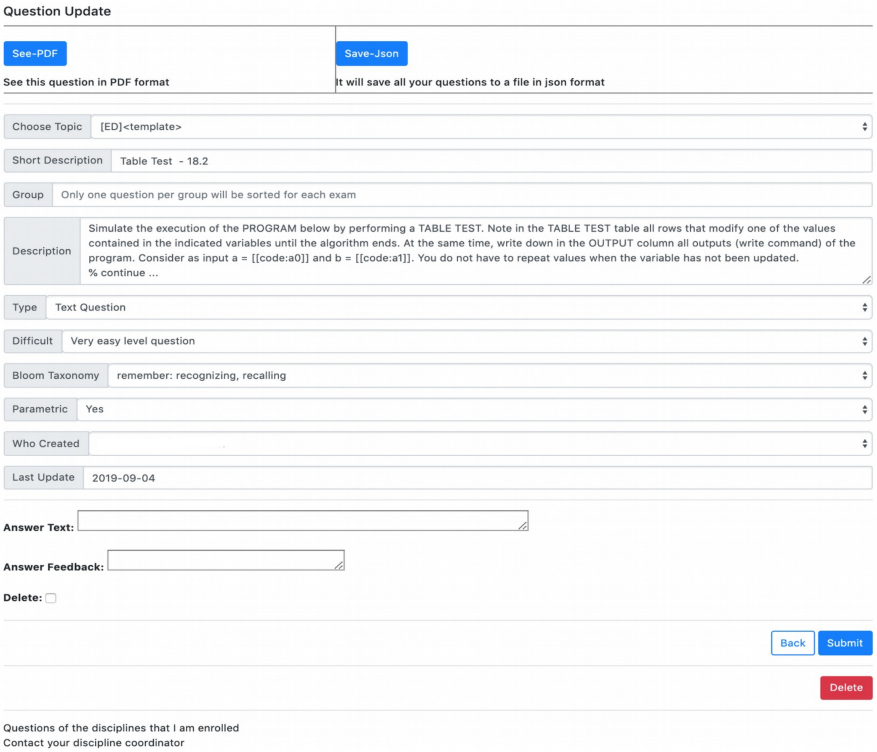
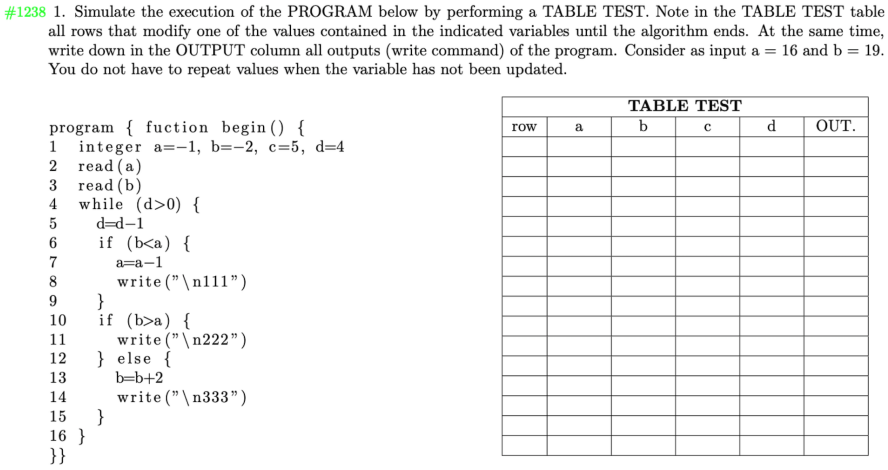


Figure 1. MCTest **screen to edit a question. This screen presents the characteristics of the Table Test question (Source: The authors).**



#1238 1. Simulate the execution of the PROGRAM below by performing a TABLE TEST. Note in the TABLE TEST table all rows that modify one of the values contained in the indicated variables until the algorithm ends. At the same time, write down in the OUTPUT column all outputs (write command) of the program. Consider as input a = 16 and b = 19. You do not have to repeat values when the variable has not been updated.

```
program { fuction begin() {
1   integer a=-1, b=-2, c=5, d=4
2   read(a)
3   read(b)
4   while (d>0) {
5     d=d-1
6     if (b<a) {
7       a=a-1
8       write("\n111")
9     }
10    if (b>a) {
11      write("\n222")
12    } else {
13      b=b+2
14      write("\n333")
15    }
16  }
}}
```

TABLE TEST

row	a	b	c	d	OUT.

Figure 2. Generated PDF by clicking the See-PDF button in Figure 1 (Source: The authors).

in online interpreters, such as `repl.it/languages/python3`, to make sure everything is correct. Initially, `myRandom` function was defined:

a. Function to create random variables: to create the 6 random variables needed in the question description. This function returns an `A` vector containing these 6 random variables, which will be the input argument of the algorithm function below;
b. Function that implements the code solution: this function algorithm implements the code solution presented in the left column of the question, see

% Part I: description of the question
Simulate the execution of the PROGRAM below by performing a TABLE TEST. Note in the TABLE TEST table all rows that modify one of the values contained in the indicated variables until the algorithm ends. At the same time, write down in the OUTPUT column all outputs (write command) of the program. Consider as input a = [[code:a0]] and b = [[code:a1]]. You do not have to repeat values when the variable has not been updated.

% Part II: table
```
\newcolumntype{C}{>{\centering\arraybackslash}p{3.1em}}
\begin{multicols}{2}
\begin{lstlisting}
```

% Part II.a: program to be simulated - Left Column
```
program { function begin() {
1  integer a=-1, b=-2, c=[[code:a2]], d=[[code:a3]]
2  read(a)
3  read(b)
4  while (d>0) {
5    d=d-1
6    if (b<a) {
7      a=a-[[code:a4]]
8      write("\n111")
9    }
10   if (b>a) {
11     write("\n222")
12   } else {
13     b=b+[[code:a5]]
14     write("\n333")
15   }
16 }
}}
\end{lstlisting}
\columnbreak
```

% Part II.b: location of Student Response - Right Column
```
\centering
\begin{tabular}{|l|C|C|C|C|c|}
\hline
\multicolumn{6}{|c|}{\textbf{TABLE TEST}} \\ \hline
row   & a   & b   & c   & d  & OUT. \\ \hline
      &     &     &     &    &      \\ \hline
% ...
\end{tabular}
\end{multicols}
\newpage
```

% Part III: the correct answer on the back of the sheet
```
{\color{bubbles}
\begin{verbatim}
[[code:mySimulation]]
\end{verbatim}
}
```

% Part IV: python code between "[[def:" and "]]"
```
[[def:
import random

# Part IV.a: function to create random variables
def myRandom():
 global a0,a1,a2,a3,a4,a5
 a0=random.randrange(11, 20, 1)
 a1=random.randrange(11, 20, 1)
 a2=random.randrange(5, 7, 1)
 a3=random.randrange(1, 7, 1)
 a4=random.randrange(1, 4, 1)
 a5=random.randrange(1, 4, 1)
 return [a0,a1,a2,a3,a4,a5]

# Part IV.b: function that implements the code solution
def algorithm(A):
 a=-1;b=-2;c=A[2];d=A[3]
 mySimulation = "a0=%d  a1=%d\n" % (A[0],A[1])
 mySimulation += "row  a   b   c  d\n"
 mySimulation += "   1   %d %d %d %d\n" % (a,b,c,d)
 mySimulation += "   2   %d %d %d %d\n" % (a,b,c,d)
 a=A[0]
 mySimulation += "   2   %d %d %d %d\n" % (a,b,c,d)
 b=A[1]
 mySimulation += "   3   %d %d %d %d\n" % (a,b,c,d)
 while d>0:
   d-=1
   mySimulation += "   3   %d %d %d %d\n" % (a,b,c,d)
   if b<a:
     a-=A[4]
     mySimulation += "   7   %d %d %d %d\n" % (a,b,c,d)
     mySimulation += "   8                          111\n"
   if b>a:
     mySimulation += "   11                         222\n"
   else:
     b+=A[5]
     mySimulation += "   13  %d %d %d %d\n" % (a,b,c,d)
     mySimulation += "   14                         333\n"
 mySimulation += str(len(mySimulation))
 return str(mySimulation)

# Part IV.c: choose A with equivalent numbers of iterations
# for test, remove "#" bellow
while True:
 A=myRandom()
 mySimulation = algorithm(A)
 if 250<len(mySimulation)<300:
   #print (A)
   #print (mySimulation)
   break
]]
```

Figure 3. Full description of the table test question presented in Figures 1 and 2 (Source: The authors).

Figure 2. This function has as input the vector `A`, with the random variables, created by `myRandom`, and returns a text in the `mySimulation` variable containing the simulation code, according to the random values stored in `A`;

c. Choose `A` with equivalent numbers of iterations: code to choose random values in `A` to have simulations stored in `mySimulation` with a similar number of characters, in this case between 250 and 300 characters, which is proportional to the number of iterations in the code to be simulated. Important: choose ranges of values in `myRandom` that do not generate an infinite loop in this part.

3.1.2. Dissertation questions: assessment with code submission on the computer

This section presents a question to be solved in the computing laboratory. The question described in this section was applied to IP–BL in Exam2, to be solved using a programming language, described in Section 4. Although IP–BL adopted Java, any programming

language supported by Moodle VPL could be used to solve the question. A version of the question in English is shown in Figure 4.

A key capability of MCTest is the support for parameterized questions. The example of the question shown here involves an operator and two vectors. The operator and the size of the vectors are parameters and can be modified to obtain several models/versions of the same question. In this example, six variations (from model A to F) were generated:

Model A. largest operator, with a vector of 20 elements;
Model B. largest operator, with a vector of 21 elements;
Model C. largest operator, with a vector of 22 elements;
Model D. smaller operator, with a vector of 20 elements;
Model E. smaller operator, with a vector of 21 elements;
Model F. smaller operator, with a vector of 22 elements.

#1239 1. Create 2 vectors $E1$ and $E2$ of integers with 22 positions each.
Read 22 elements by storing them in the $E1$ vector.
Fill in the vector $E2$ from $E1$ based on the following rule, while k is the index variable that will be used to access both vectors:

- if $k = 0$, $E2[k]$ receives smaller elements into $\{E1[21], E1[0], E1[1]\}$;
- if $k = 21$, $E2[k]$ receives smaller elements into $\{E1[20], E1[21], E1[0]\}$;
- if k its between 1 anb 21, that is, $1 \leq k < 21$, $E2[k]$ receivessmaller elements into $\{E1[k-1], E1[k], E1[k+1]\}$.

ATTENTION:
Submit the file **exam.java** (with the answer) and the file **model.txt**, containing only the text **Model: F**. First upload the java file and then the txt file.

Example:

```
input : 1 8 6 0 8 6 4 8 7 4 3 8 7 6 4 2 2 7 3 8 3 0
output: 0 1 0 0 0 4 4 4 4 3 3 3 6 4 2 2 2 2 3 3 0 0
```

Figure 4. Automatically generated example for a dissertation question. The values drawn in the question description should yield similar examples. In IP–BL, practical Exam2, different models were generated for each question. In this example, Model F has been drawn. The student must submit their Moodle VPL solution in the corresponding template (Source: The authors).

The variation of the operators automatically updates the input and output example shown to the student (Figure 4). It is up to the teacher to elaborate parameterized questions, defining which values can be chosen for each variation to maintain the same level of difficulty among the models/versions.

To provide the students with automated feedback on the questions in the laboratory, the answers could be submitted to the Moodle VPL plugin (Rodríguez-del Pino et al., 2012). Since there are several models/versions of each question, the student had to submit a file containing the model identifier (letter) of the question together with the answer. Based on that, the test cases file is selected. An example of the format of test cases is shown in Figure 5.

The following are all the details of the question in Figure 6. The red text is the parameterized part of the question. This figure has several parameter values, it can become complex. A more detailed description of the question is presented below:

Part I. Description of question: Displays the description that will appear on the student exam. In the first paragraph, two parametric values were defined, `var1` and `a0`. The first assumes a random letter between `A` and `G`. Variable `a0` assumes a random

case=test1
input = 6 1 5 8 3 5 5 8 5 2 1 8 7 7 3 4 5 0 6 2
output= 6 6 8 8 8 5 8 8 8 5 8 8 8 7 7 5 5 6 6 6
case=test2
input = 0 2 5 5 1 6 4 4 3 0 3 8 2 8 3 7 1 0 8 8
output= 8 5 5 5 6 6 6 4 4 3 8 8 8 8 8 7 7 8 8 8

Figure 5. Test cases to be used by Moodle VPL for Model F (Source: The authors).

% Part I: description of the question
Create 2 vectors $ [[code:var1]] 1$ and $ [[code:var1]] 2$ of integers with $[[code:a0]]$ positions each.

Read $[[code:a0]]$ elements by storing them in the $ [[code:var1]] 1$ vector.

Fill in the vector $[[code:var1]] 2$ from $ [[code:var1]] 1 $ based on the following rule, while $ [[code:var0]] $ is the index variable that will be used to access both vectors:

\begin{itemize}
\item if $ [[code:var0]] = 0$,
$[[code:var1]] 2[[[code:var0]]]$ receives **[[code:a1]]** elements into $\{[[code:var1]] 1[[[code:a0-1]]], [[code:var1]] 1[0], [[code:var1]] 1[1]\}$;

\item if $ [[code:var0]] = [[code:a0-1]]$,
$[[code:var1]] 2[[[code:var0]]]$ receives [[code:a1]] elements into $\{[[code:var1]] 1[[[code:a0-2]]], [[code:var1]] 1[[[code:a0-1]]], [[code:var1]] 1[0]\}$;

\item if $ [[code:var0]]$ it's between 1 anb $[[code:a0-1]]$, that is, $1\leq [[code:var0]] < [[code:a0-1]]$,
$ [[code:var1]] 2[[[code:var0]]]$ receives[[code:a1]] elements into $\{ [[code:var1]] 1[[[code:var0]] -1], [[code:var1]] 1[[[code:var0]]], [[code:var1]] 1[[[code:var0]] +1]\}$.
\end{itemize}

\noindent\textbf{ATTENTION:} \\ Submit the file \textbf{exam.java} (with the answer) and the file \textbf{model.txt}, containing only the text \textbf{Model: **[[code:model]]**}. First upload the java file and then the txt file.

% Part II: the correct answer, like an example
\vspace{5mm}\noindent\textbf{Exemplo:}
\begin{verbatim}
[[code:mySimulation]]
\end{verbatim}

% Part III: python code between "[[def:" and "]]"
[[def:
```
import random, numpy as np

# Part III.a: create random variables
a_tam   = 2 # random.randrange(0,3,1)
a_inicio = 20
a0=random.randrange(a_inicio, a_inicio+a_tam+1, 1)
oper = ["larger","smaller"]
a1=random.choice(oper)
letters = ["A","B","C","D","E","F","G"]
model = letters[oper.index(a1)*(1+a_tam)+(a0-a_inicio)]
var0 = random.choice(["i","j","x","y","w","k","p"])
var1 = random.choice(letters)
global mySimulation
v1 = np.random.randint(9, size=a0)
v2 = np.zeros(a0, dtype='int')

# Part III.b: implements the code solution
mySimulation='input : '+ ' '.join([str(i) for i in v1])+'\n'
for i in range(a0):
 aux = [v1[(i-1)%a0], v1[i], v1[(i+1)%a0]]
 if a1==oper[0]: #max
   v2[i]= np.max(aux)
 if a1==oper[1]: #min
   v2[i]= np.min(aux)
mySimulation += 'output: ' + ' '.join([str(i) for i in v2])
```
]]

Figure 6. Full description of the dissertation question (Source: The authors).

value between 20 and 22 (representing the size of the vectors). Thus, in Figure 4 were drawn the values `E` and `22`, respectively. Another important variable that appears in the description of the question is `a1`, which can assume the values "larger" or "smaller". The central part of this question is to create examples and choose a question model, as follows;

Part II. The correct answer, like an example: Formats the example, containing the question inputs and outputs, drawn and calculated in Part III;

Part III. Python code between "`[[def:`" and "`]]`": This part contains Python code to provide the contents of the variables that appear in the question description, as presented in the previous parts:

a. Create random variables: Note that, unlike Subsection 3.1.1, no function has been defined in this description to create the variables that appear in the question description. In addition to the variables already mentioned in Part I, a random vector `v1` was created as input and the output vector `v2`, which must be calculated using the code solution. The content of `v1` was obtained with the command `v1=np.random.randint(9,size=a0)`, generating a vector `v1` of random elements `a0`, see Figure 5;

b. Implements the code solution: This part presents the solution of the problem implemented in Python, however, with the concern of formatting the

input and output vectors.

3.2. Proposal to integrate MCTest and VPL – applied in 2019

The main contribution of this article is the use of parameterized questions with automated assessment using the VPL plugin (Rodríguez-del Pino et al., 2012). MCTest produces parameterized questions, as described in the previous sections. However, a major difficulty for teachers in programming courses with many students is correction and feedback. The VPL plugin came to provide this part, however, so far, without addressing the parametric part. To solve this, this section describes how we can automate the parameterized Programming Exercise (PE), which is the main contribution to the state of the art of Information and Communication Technology (ICT) in education presented in this article.

As the default VPL question cases file accepts only a single version of a question, it would be necessary to create a PE for each version of every question in an exam. For example, an exam with three questions, each with six variations, the teacher would need to create and configure 18 different PE. Then, the students would have to choose the correct versions according to the exclusive exams, something that could lead to confusion and mistakes.

Using the modified system, the teacher needs to create only 3 PE with 6 internal version files. Then, the student needs to submit a second text file containing a single line as "`MODEL: A`" which is the model/version of the question. This text file will be parsed using a regular expression, isolating the version/model code for that student, and then be used to choose the correct version of the test case files used by the teacher. Every version X has its own `vpl_evaluate_X.cases` file, which replaces the default VPL `vpl_evaluate.cases` file.

After a Moodle VPL activity is created, in the runtime files, the teacher would have to add six test cases and the other files available at `github.com/fzampirolli/mctest/VPL_modification/V1-select_using_second_file`. These files were used for the question presented in Figure 4 and applied to Exam2's first question in the two IP–BL classes in 2019.1.

4. Preliminary results and discussions

Introduction to programming (called *Processamento da Informação*) is a 12-week course of the Bachelor of Science and Technology at the Federal University of ABC. In the first four months of 2019, it received 1,437 students enrolment divided into 46 laboratory or blended learning classes (IP–BL). Laboratory adopted Java (36/46=78% of classes) or Python (8/46=17%) programming languages. In contrast, IP–BL classes applied a pseudocode-to-Java combination (2/46=4%), i.e., students developed pseudocode solutions and translated them (through Portugol Studio tool - available at `lite.acad.univali.br/portugol`) to Java code. The focus of this section is on IP–BL where our proposal was applied in 2019.

Figure 7 shows a summary of the distribution of the grades obtained by the students over the last years in IP–BL. The final grade is the result of Formative Assessments (18 Multiple-Choice Tests = 5% and 17 PE = 5%), Exam1 (35%), Group project of 3 students (15%) and Exam2 (40%). However, there were some differences among the assessment criteria in some periods. For example, in periods 18.3, 18.2, and 17.3, the

student was required to take a recovery exam if the grade in Exam2 were F. Moreover, it must also be noted that in 19.1, 18.1, 17.2, and 17.1 additional motivational messages were sent to the students. These findings suggest that distinct evaluation methods can impact students' grades and adopting uniform rules for assessments may avoid such a grade variation.

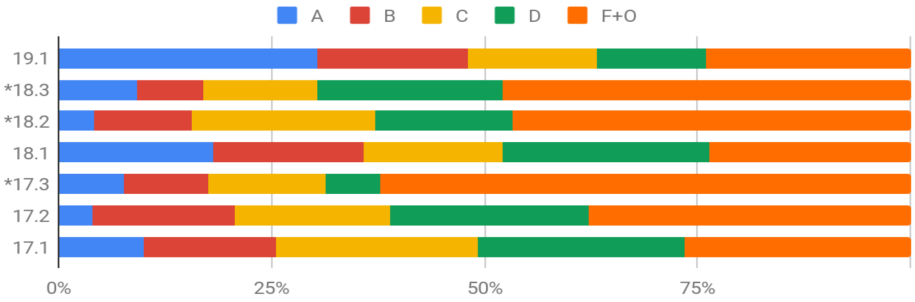


Figure 7. IP–BL performance in recent years. A high failure rate is observed in periods 18.3, 18.2, and 17.3, when an Exam2 = "F" rule was used, so that the student was required to take the recovery exam. The grade "O" means failure due to a high number of absences in a class (Source: The authors).

Despite these differences among the periods, we observed an overall improvement in the distribution in 19.1 when AA was applied. However, we cannot say that this is the reason for the improved performance. Perhaps, the fact of adopting AA was one of the reasons for the observed results. This analysis is not conclusive and requires a long-term study. To exemplify this need, class A1 (with 116 students) obtained 86.21% approval, while class A2 (with 55 students) obtained 54.55%. Both IP–BL classes had the same teachers, content, exams, and evaluation criterion. A possible justification for this difference in results between classes is that class A1 was taught in Santo André and A2 in São Bernardo do Campo, with different student profiles on each campus.

5. Conclusions and future works

The discussion presented in this paper is evidence that it is important to join efforts in Information and Communication Technology (ICT) development to help teachers fairly and efficiently evaluate several students at the same time, creatively using MCTest+Moodle+VPL exams. The MCTest was adopted in this article to elaborate parameterized questions, including examples that were used in the VPL activities in Moodle, such as test cases for Automated Assessment (AA). This process was used in two blended learning classes (a total of 171 students), with activities in the Java programming language.

A key advantage of our proposal is the automated generation of several models/versions of exams and its integration between MCTest and VPL plugin in Moodle. This is useful to avoid that the students share solutions since the exams would not be exactly the same. This feature can be particularly important for exams applied remotely.

Finally, after the 2019.1 offer, we have worked on an improvement of our proposal to generate individualized exams for each student, with AA in the VPL, without the need to submit a file informing the question model/version. This improvement is currently under validation and may be discussed in the future.

References

Alemán, J. L. F. (2010). Automated assessment in a programming tools course. *IEEE Transactions on Education*, 54(4):576–581.

Bez, J. L., Ferreira, C. E., and Tonin, N. (2013/08). Uri online judge academic: A tool for professors. In *Proceedings of the 2013 International Conference on Advanced ICT and Education*, pages 744–747. Atlantis Press.

de Campos, C. P. ; Ferreira, C. E. (2004). Boca: A support system for programming contests. In *Brazilian Workshop on Education in Computing*.

Demir, Ö., Soysal, A., Arslan, A., Yürekli, B., and Yılmazel, Ö. (2010). Automatic grading system for programming homework. *Computer Science Education: Innovation and Technology, CSEIT*.

DuFrene, A. (https://digitalcommons.calpoly.edu/cscsp/95 (accessed June 23, 2020)). *Automatic Generation and Grading of Programming Exercises*.

Galan, D., Heradio, R., Vargas, H., Abad, I., and Cerrada, J. A. (2019). Automated assessment of computer programming practices: The 8-years uned experience. *IEEE Access*, 7:130113–130119.

Gordillo, A. (2019). Effect of an instructor-centered tool for automatic assessment of programming assignments on students' perceptions and performance. *Sustainability*, 11(20):5568.

Hagiya, M., Fukuda, K., Tanabe, Y., and Saito, T. (2019). Automatically generating programming questions corresponding to rubrics using assertions and invariants. In Tatnall, A. and Mavengere, N., editors, *Sustainable ICT, Education and Learning*, pages 89–98. Springer International Publishing.

Radošević, D., Orehovački, T., and Stapić, Z. (2010). Automatic on-line generation of student's exercises in teaching programming. In *In Central European Conference on Information and Intelligent Systems, CECIIS*.

Rodríguez-del Pino, J. C., Rubio Royo, E., and Hernández Figueroa, Z. (2012). A virtual programming lab for moodle with automatic assessment and anti-plagiarism features. *Conference: International Conference on e-Learning, e-Business, Enterprise Information Systems, & e-Government*.

Rubio-Sánchez, M., Kinnunen, P., Pareja-Flores, C., and Velázquez-Iturbide, Á. (2014). Student perception and usage of an automated programming assessment tool. *Computers in Human Behavior*, 31:453–460.

Staubitz, T., Teusner, R., and Meinel, C. (2017). Towards a repository for open autogradable programming exercises. In *2017 IEEE 6th International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, pages 66–73. IEEE.

Venero, M. F. and Mena-Chalco, J. (2019). Ensino de programação avançada incentivando a metacognição: uma experiência positiva usando Moodle+VPL. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 30, page 279.

Wang, T., Su, X., Ma, P., Wang, Y., and Wang, K. (2011). Ability-training-oriented automated assessment in introductory programming course. *Computers & Education*, 56(1):220–226.

Zampirolli, F., Teubl, F., and Batista, V. (2019). Online generator and corrector of parametric questions in hard copy useful for the elaboration of thousands of individualized exams. In *CSEDU (1)*, pages 352–359.