A Generator and Corrector of Parametric Questions in Hard Copy

Francisco de Assis Zampirolli, Fernando Teubl, Valério Ramos Batista
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC (UFABC)
9.210-580 – Santo André – SP – Brazil
{fzampirolli,fernando.teubl,valerio.batista}@ufabc.edu.br

***Abstract*—The demand and use of automatic test generators for exams and competitions have both been increasing. But the handiness offered by these generators brings a difficulty in the process of creating databanks of questions: from a group of questions their different versions are still made for each question separately. Of course, this manual task turns out to be lengthy and costly. This problem proved to be effectively solved by generators of parametric questions. Our present work introduces an innovative and accessible approach that consists of an open platform to generate and correct parametric questions. To the best of our knowledge the platform presented here is the first one totally devoted to exams in hard copy that also includes automatic correction. Teachers and professors of any institution that registers in our platform can share question databanks among them.**

***Index Terms*—parametric questions; test generators; databanks of questions; GUI; Python.**

I. Introduction

Test generators have been easing the arduous tasks of elaborating and correcting numerous questions for exams given by teachers and panels. Usually these generators make tests automatically according to a databank of questions and predefined criteria, so that each student sits the exam with the same content but in a different version.

A test generator can include at least one of the following steps: (a) performing a random choice of questions from a databank, (b) making different versions of a question either in its statement (*parametric*) or in the order of answers (*multiple-choice*), (c) formatting and consolidating individual tests, and (d) following automatic methods to correct the tests, these either online or in hard copy.

Steps (a) and (c) are relatively simple, whereas (b) and (d) are pretty challenging. For instance, in (b) the user is frequently a teacher who must establish well-defined criteria upon which the question can be changed, so that the different answers are achieved with the same difficulty. Such parametrizations may require technical skills the users do not have unless they undergo some training to elaborate questions, mainly of the sort that fits a specific course. This is a typical requirement that risks turning inviable the popularisation of parametric question generators.

Regarding (d), on the one hand it is simple for tests carried out electronically. On the other hand, for handwritten tests it turns out to be a laborious step. The automatic corrector has to interpret and recognize the answers through computer vision and compare them with the answer keys. Usually each version of the test corresponds to a different answer key.

This paper introduces one of our contributions to the research area of Education. Herewith we present an automatic generator and corrector of random tests with parametric questions, which is devoted to exams in hard copy. Our programs were implemented in Python, together with open-source libraries, and they are applicable to large classes that consist of students in hundreds. Of course, for such a class any exam requires efficient processes of massive test generation and correction. Our present work contributes to meet the increasing demand for massive test generators that also ensure the integrity of the students' marks. There are several other similar generators in the literature but herewith we propose an accessible and free-of-charge program that also includes a new method to encode parametric questions.

II. Related Works

There are several test generators in the present day. Some of them include built-in tools for parametric questions and automatic correction of tests. As mentioned at the Introduction these features are already implemented in our proposal but we not only strive for a user-friendly platform with graphical interface: our program can be used for free and the encoding of the answer keys is inviolable.

Now we briefly describe some of these generators.

The *QuizPACK*[1] is a system developed in C programming language for both production and correction of questions. Questions must be parametrized by the teachers themselves directly in C language. Afterwards they have to upload their program files into the system. Parameters are identified through a pseudo-variable Z in the code, and then steadily replaced with random integers while generating the tests. These integers range in an interval given by the user [1], [2].

In [3] the author describes his *Matlab Implementation of the Automatic Generator of the Parameterized Tasks*. He developed a self-sufficient structuring in XML to write parametrized questions. There one can generate questions whose answers may be numerical, phrasal, multiple-choice or cloze. The generating process begins with the uploading of a text whose

[1] *Quizzes for Parameterized Assessment of C Knowledge.*

parameters are marked with *tags*, namely <input> containing both name and type of the variables [3]. After uploading the questions in XML the generator will produce the tests with their parametrized questions by attributing random values to these variables within a certain interval. The final output is also written in XML but can be re-formatted, for example into *Moodle XML* or LaTeX [3].

SmartQuestion (sQ) is a software that automatically generates questions with multiple-choice answers. It follows the concept of "a way to store all static versions of a question in the same parametric question". This tool offers three tabs: sQd (*sQ design*), sQp (*sQ parameterize*) and sQg (*sQ generate*) [4]. The software sQ starts with the *bitmap* or *JPEG* image of the static question given by the users. With sQd they can define a place for each parameter, which may be either in the statement of the question (regular) or in the answer (for the alternatives). With sQp the users select parameters and insert new values in a textbox compatible with LaTeX. Then sQg makes sub-images of the changed parameters and incorporates them into the respective places. The final result is a complete image of the new question. With sQg one can also shuffle the alternatives so that versions vary in both statement and order of answers [4].

MEGUA (***M**athematics **E**xercise **G**enerator, **U**niversidade de **A**veiro*)[2] is an open-source software that enables one to make databanks of parametrized questions with their respective answers all in LaTeX. It works with the mathematical software *SageMath*, which uses Python programming language [5]. Its question databanks are called "*Books*" and built through either PDFLatex (for hard copy) or HTML and MathJAX (for web publishing) [6]. The elaboration of a question essentially occurs inside the *Notebook* of *SageMathCloud*, whose name has changed to *CoCalc* since May 2017. This one consists of three steps: Firstly, on a new *worksheet* we make a cell to import the whole MEGUA library and open/create a databank in which the questions will be stored. Secondly, the code of the question is introduced into another cell that consists of a text in LaTeX and the programming in Python. The LaTeX part is divided in sections (cataloguing and description of the exercise), "%problem" (name and statement of the question) and "%answer" (its solution). Finally *CoCalc* is concluded by the programming part, which contains two functions: "make_random" (generates random values for the statement) and "solve" (computes the right solution and gives others for the multiple-choice). The output of this cell execution is two files, one in PDF and other in TeX [7]. There is also a resource to add parametrized graphs to the exercises. However, MEGUA is not endowed with automatic correction of parametric questions in hard copy, a desirable feature at evaluating hundreds of candidates.

TestMakPro3 is a commercial software that generates parametric questions. It was implemented in *JavaScript* [8] and enables the user to elaborate the statement of a question with at most three variables, together with their range of values. For each variable we set its margin of error and its number of decimals to be given in each answer.

AMC is a free software to produce and manage multiple-choice quizzes, which also performs their automatic correction. Tests are written in LaTeX but any user unfamiliar with this document preparation system can give them written in a special format called *AMC-TXT*, whose syntax is quite simple. One of the facilities of *AMC* is the option to shuffle questions so that each student's version turns out to be unique. Another is an automatic correction of the scanned tests that already brings their marks [9], [10]. But *AMC* does not work with parametric questions. Moreover, though *AMC* includes a database of students the tests are first associated to them only in the correction process, hence each student must fill out their identification data. Corrections are carried out inside the very statements of the questions, so that one has to digitize all exam pages for the correction process.

MakeTests[3] is an open-source program written in Python. Its main purpose is to generate tests out of a totally random choice of questions. For this purpose *MakeTests* enables the user to write questions programmed in Python and to classify them through directories. These correspond to subjects and levels of difficulty. *MakeTests* writes the exam headers and the random questions in PDF through specifications described in a configuration JSON file. The most important feature of *MakeTests* is the production of questions in a totally independent way, since one writes each question directly in Python (interfaced with LaTeX). The user can set any type of answers: multiple-choice, true/false, numerical, etc. The greatest difficulty in *MakeTests* is that the user must have advanced knowledge of Python to programme questions, and this is impracticable in several areas of Education. Moreover, *MakeTests* does not include automatic correction: it generates answer keys but the correction is performed manually.

In general all approaches in the literature are somehow efficient at tackling production and/or correction of tests. However, their main common point is the inaccessibility either because the users have to master programming languages (eg. [1], [6]) and/or to pay for the software (eg. [3], [8]). Table I summarizes this section, together with our proposal MCTest [4], whose previous versions could only circumvent the absence of parametric questions, as explained in Sect. III-A, and also a mechanism for them that produces alternatives without repetition, detailed in Sect. III-B. Our software present day's version is called MCTest 4.1.

III. Methods

Our method to produce parametric questions is now presented here. Firstly we describe the MCTest platform in the standard configuration. Afterwards we explain further implementations that now bolster the new functionalities.

[2]The name "MEGUA" stands for a trademark of the University of Aveiro since 2012 [5].

[3]Available at https://github.com/fernandoteubl/MakeTests

[4]Available at https://github.com/fzampirolli/MCTest4

	QuizPack	[3]	SmartQuestion	MEGUA	TestMakPro3	AMC	MakeTests	MCTest
Language	C	Matlab		Python	JavaScript	Perl	Python	Python
Commercial					×			
Open-Source						×	×	×
Parametric Questions	×	×		×	×		×	×
Automatic Correction of Hard Copies						×		×
User Has to Programme	×	×		×			×	†

†: Only if you make use of parametric questions.

TABLE I
COMPARISON BETWEEN TEST GENERATORS

A. MCTest

1) History: Nowadays the version 4.0 of our free software MCTest is finally available. MCTest has started in 2010 to automate an exam for openings in a specialization course at a Brazilian university. More than thousand candidates enrolled for that exam.

In the present day we apply MCTest to produce and correct over ten thousand tests every year. Version 1 was implemented in Matlab with automatic correction performed by snapping the exams with a computer webcam. Until version 3 the exams were written in traditional text editors. Version 2 was implemented in Java for Android. The 3rd version used Python, still only for correction of tests, which were digitized as input. Finally in the version 4.0 we added production of tests in Python through LaTeX.

In the present version multiple-choice tests are automatically corrected by uploading the answer cards digitized in PDF. This must be done via the ftp server vision.ufabc.edu.br by following the steps explained in a previous publication.

2) Present Day's Version of MCTest: MCTest can produce exams corresponding to students of given classes. The students' data must be given by CSV spreadsheet, and questions are taken from a databank in which they must be written according to a specific template in TXT format (see Fig. 1).

In Fig. 1 we see five multiple-choice questions written in a TXT file. Their levels of difficulty are assigned as QE (easy), QM (medium) and QH (high). For instance, in Fig. 1 there are three easy questions, one medium and one difficult. For multiple-choice questions each alternative must begin with "A:", and MCTest will always take the 1st one to compose the whole answer key. Of course, MCTest performs a nested shuffling: of questions and of their *attached* alternatives. The user can also include dissertation questions by toggling them with "QT". MCTest produces all exams written in LaTeX with a custom header. A resulting PDF can be seen in Fig. 2, already with the nested shuffling. In Fig. 1, when the same character appears after the phrase `topic #::`, for instance "a::", this means that for each student only one of the two questions in Fig. 1 will be drawn.

When we just want to change some values manually, either in the statement or in the alternatives of a question, and then perform a random choice of them for each student,

```
QE::topic 1:: Question Q1-example of equation:
$\sin A \cos B =
\frac{1}{2}\left[ \sin(A-B)+\sin(A+B) \right]$
A: answer 1a
A: answer 1b
A: answer 1c
A: answer 1d
A: answer 1e

QE::topic 2::a:: Question Q2
A: answer 2a
A: answer 2b
A: answer 2c
A: answer 2d
A: answer 2e

QE::topic 2::a:: Question Q3
A: answer 3a
A: answer 3b
A: answer 3c
A: answer 3d
A: answer 3e

QM::topic 3::  Question Q4
A: answer 4a
A: answer 4b
A: answer 4c
A: answer 4d
A: answer 4e

QH::topic 4:: Question Q5
A: answer 5a
A: answer 5b
A: answer 5c
A: answer 5d
A: answer 5e
```

Fig. 1. Example of questions written for MCTest.

the model presented in Fig. 1 is inefficient. This is because duplicating and then changing a text manually can lead to errors. Another resource not available in MCTest 4.0 is the automatic computation of the right answer, which should then be included as one of the alternatives.

These are the main reasons for us to carry on version 4.1 of MCTest, which includes methods to generate parametric questions and to compute the right answer, together with the

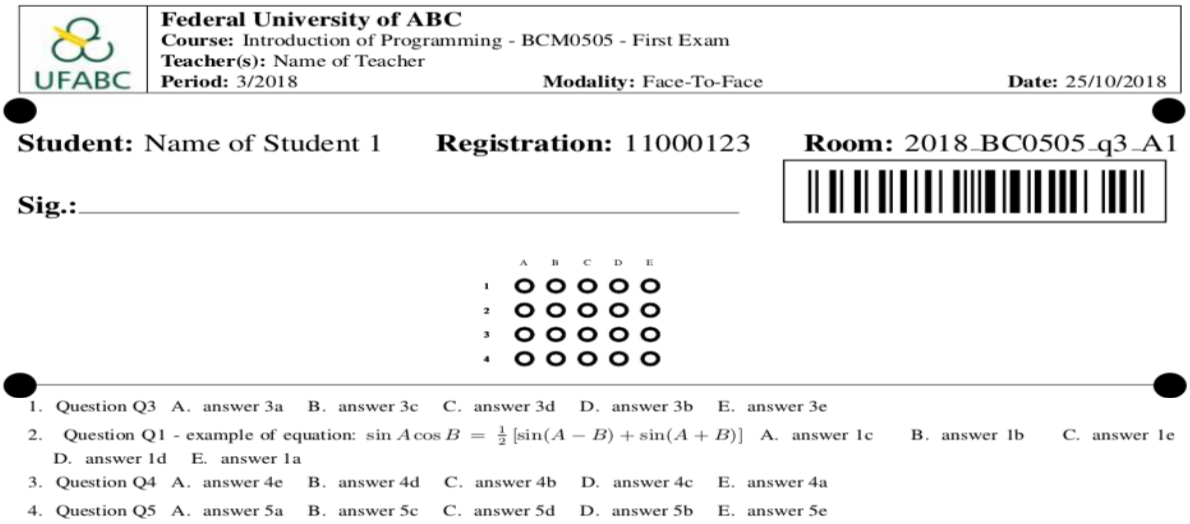

Fig. 2. A test generated by MCTest.

automatic production of false alternatives, as we are going to see in the next subsection.

B. MCTest with Parametric Questions

In order to implement parametric questions in MCTest we had to include two special delimiters `[[code: ...]]` and `[[def: ...]]`. The former must come in the statement, either of the question or of the alternatives. With `[[code: ...]]` we define the parameters (or variables) to be used at generating each student's test. The latter must be defined at the end of the TXT file. In this case each parametric question must be in a separate file. Moreover, `[[def: ...]]` has to contain the possible values applied beforehand, together with a method called `algorithm`. In the next section we show some examples of parametric questions included in MCTest 4.1 by means of these delimiters.

IV. Results

By the methods just presented in the previous section, now we illustrate how they solve two practical problems that involve parametric questions.

A. Parametric Question for ULM

Uniform Linear Motion (ULM) is a typical subject of parametric questions. In Fig. 3 we see a complete example for a Physics exam written in TXT.

On the first line we classify both the difficulty and the subject `QE::ulm::`. Notice that we may use LaTeX syntax in the statement and in the alternatives. For instance, comments are toggled with % and math symbols enclosed with $. For a multiple-choice question, right after the statement we include each alternative followed by `A:`. The first must be the right one, according to the MCTest method. Therefore we have `A: [[code:correctAnswer]]`, where `correctAnswer` is a variable computed in `[[def:`, which also defines a function called `algorithm`.

```
QE::ulm:: % question text
A car moves on a road with an hourly function
$s=[[code:a0]] + [[code:a1]]t$, where $s$ is
given in miles and $t$ in hours. The car passes
the mile [[code:a2]] exactly at:

% answers
A: [[code:correctAnswer]]
A: [[code:correctAnswer-1]]
A: [[code:correctAnswer-2]]
A: [[code:correctAnswer+1]]
A: [[code:correctAnswer+2]]

[[def:
a0 = random.randrange(-6, 3, 1) # return
a1 = random.randrange(3, 8, 1)  # random
a2 = random.randrange(3, 8, 1)  # numbers

def algorithm(a):
    from sympy import *
    a0=int(a[0])
    a1=int(a[1])
    a2=int(a[2])
    s,t = symbols('s,t')
    s=a0+a1*t
    r = float(solve(s-a2,t)[0])
    return r

global correctAnswer
correctAnswer= algorithm([a0,a1,a2])
]]
```

Fig. 3. Example of a ULM question.

The algorithm must have all variables in the statement as parameters. The ULM problem asks for the solution of an algebraic equation. For that we use the Python library *sympy* and define two variables: `s` and `t`. Afterwards one simply writes the corresponding equation `s=a0+a1*t`. The result is given by the function *solver* of the library *sympy*. Three

random variables `a0`, `a1` and `a2` come either in the statement or in the alternatives. That is why we have included the function *random.randrange* from Python.

Finally we must assign `algorithm([a0,a1,a2])` to the global variable `correctAnswer` in order to make it store the correct result, which is then inserted in the first alternative. The wrong alternatives were defined as the solution ± 1 and ± 2. These are arbitrary values but most importantly is not to repeat alternatives. We can resort to another method that produces wrong answers and avoids repetition, as shown in the next example. Fig. 5 depicts three random outputs of that ULM problem.

B. Parametric Question to Handle Matrices

Fig. 4 illustrates an example for handling matrix to evaluate the students' logical programming skills in a chosen language. Dimensions are large because the students have to solve through a program code. They must compute the sum of the entries of a matrix whose dimensions and elements are both randomly taken while MCTest produces each student's test. Differently from the previous question, this one resorts to a new method called `createWrongAnswers([5,10])`, which produces five distinct random values between `correctAnswer-10` and `correctAnswer+10`. Fig. 4 shows the text in TXT that made MCTest produce the three outputs in Fig. 6.

C. Experiments

Our new method to generate parametric questions was applied for the course Introduction to Computer Science (ICS) at our university in the second trimester of 2018. On that occasion we had 167 matriculated students and ICS was a half-distance-learning half-classroom course. In the classroom their presence took place 4 times: opening, first test, project and second test. This course belongs to an interdisciplinary bachelor's programme offered by our university, where the students come from many different backgrounds.

Our university's instruction periods are organized in three trimesters. Here all undergraduate students follow two Interdisciplinary Bachelor's Programmes (IBP): Science&Technology and Science&Humanities abbreviated as BCT and BCH, respectively. On average we have 1,600 freshmen in BCT and 400 in BCH annually. ICS is mandatory for all students, it belongs to BCT and is always scheduled for the third trimester.

Both BCT and BCH take three years, and with an additional year the student can graduate either in Maths, in Physics, or in Computer Science. Right after BCT, with two additional years the student can graduate in one of the nine engineering programmes offered by our institution. Since our students come from many different backgrounds, ICS is devoted to teaching logic programming for daily problems on several subjects in the student's life.

In the classroom we gave tests with parametric questions. The first exam was in hard copy with dissertation questions. The project and the second exam were given in a computer laboratory, where students had to hand in three program codes

```
QE::matrix:: % question text
Build a matrix $[[code:a0]] \times
[[code:a1]]$ whose elements $(i,j)$ are
$((((i+1) * [[code:a2]]) + ((j+1) *
[[code:a3]])) \mod{100})$. Compute the sum
of the entries of this matrix. Indexes $i$
of rows and $j$ of columns begin with $0$.

% createWrongAnswers([5,10]) - Makes
% 5 different wrong alternatives
% between +/- 10
A: [[code:correctAnswer]]
A: [[code:createWrongAnswers([5,10])]]

[[def:
# code to return a correct answer with the
# following variables
def algorithm(a):
  a0 = int(a[0])
  a1 = int(a[1])
  a2 = int(a[2])
  a3 = int(a[3])
  P = np.zeros((a0,a1))
  for i in range(a0):
    for j in range(a1):
      P[i,j] = (((i+1)*a2)+((j+1)*a3))%100
  return int(P.sum())

# variables used in the question
# produces a random number between 60 and 80
a0=random.randrange(60, 80, 1)
a1=random.randrange(60, 80, 1)
# takes a number at random from a set of three
a2=random.choice([7, 13, 19])
a3=random.choice([11, 17, 23])

global correctAnswer
correctAnswer= algorithm([a0,a1,a2,a3])
]]
```

Fig. 4. Example of a parametric question on matrix.

by uploading them into a virtual learning platform. Each exam consisted of three questions: of easy, medium and high difficulty. Moreover, each question type came in different versions, as depicted in Figs. 4 and 6. Notice that the versions do not change the logic programming for solving them.

Hence, MCTest enables us to produce several versions of the same question without repetition. We just need one statement as shown in Figs. 4 and 6. Thus, for each student a different triple of questions was generated (of easy, medium and high levels of difficulty). Some examples of easy level are illustrated in Fig. 6.

V. Conclusions

The main contribution of our work is a method to generate parametric questions for exams in hard copy. Both production and generation of multiple-choice questions are free softwares, and the former even open-source. Automatic correction of the multiple-choice answer cards follows the method presented in a previous work. To the best of our knowledge MCTest is the first platform totally devoted to exams in hard copy that

1. A car moves on a road with an hourly function $s = 1 + 5t$, where s is given in miles and t in hours. The car passes the mile 3 exactly at:
   A. -0.6 B. 0.4 C. 2.4 D. -1.6 E. 1.4
2. A car moves on a road with an hourly function $s = 1 + 4t$, where s is given in miles and t in hours. The car passes the mile 7 exactly at:
   A. 0.5 B. 3.5 C. 1.5 D. 2.5 E. -0.5
3. A car moves on a road with an hourly function $s = -3 + 6t$, where s is given in miles and t in hours. The car passes the mile 3 exactly at:
   A. -1.0 B. 2.0 C. 3.0 D. 1.0 E. 0.0

Fig. 5. Example with three outputs of the parametric ULM.

1. Build a matrix 68×75 whose elements (i, j) are $((((i+1)*19)+((j+1)*11)) \mod 100)$. Compute the sum of the entries of this matrix. Indexes i of rows and j of columns begin with 0.
   A. 252348 B. 252350 C. 252349 D. 252354 E. 252346 F. 252355
2. Build a matrix 74×74 whose elements (i, j) are $((((i+1)*19)+((j+1)*11)) \mod 100)$. Compute the sum of the entries of this matrix. Indexes i of rows and j of columns begin with 0.
   A. 270897 B. 270891 C. 270898 D. 270893 E. 270900 F. 270896
3. Build a matrix 67×77 whose elements (i, j) are $((((i+1)*19)+((j+1)*23)) \mod 100)$. Compute the sum of the entries of this matrix. Indexes i of rows and j of columns begin with 0.
   A. 255332 B. 255338 C. 255331 D. 255337 E. 255346 F. 255327

Fig. 6. Example of three outputs from the matrix parametric question.

includes automatic correction, besides production of tests with random answer keys.

In this paper we explained how to produce parametric questions with MCTest. Though it apparently requires some knowledge of both LaTeX and Python, MCTest offers a folder with several templates that can be used by any teacher and any professor, and they just have to adapt the templates to write their personal questions. Moreover, the MCTest exam generator is open-source, and the several examples of parametric questions available in our repositories can not only help teachers and professors check what we have already done, but they can also contribute with new models of questions. With this interaction we hope to achieve a great facilitation in the arduous task of evaluating large classes.

With MCTest we can also give weekly tests without much effort. This helps track the gradual performance of each student through periodical and personal evaluations, as suggested by Gusev et al. [11]. For example, if a student cannot answer questions about a certain subject, then we include more related to questions in the following exam. Our approach also differs from others in the sense that we focus on exams in hard copy, which include parametric questions instead of quizzes on the web (Gasev's proposal does not include parametric questions either).

In this paper we only gave examples of mathematical and programming questions but an online version of MCTest is under development and it will include a great variety of parametrized questions. But the teacher/professor will always have to prepare them in order to make a reasonable test that covers the programme of the course, with an adequate number of questions, alternatives per question and so on. For this task [12] presents many results of preparing questions based on cognitive model.

In a future work we shall implement a web server endowed with access control, for teachers and professors to store their parametric questions through a friendly Graphical User Interface (GUI). Parametric questions will also take other formats besides multiple-choice. For instance, dissertative and true/false models. Besides Python, solution of parametric questions by other programming languages could be easily encompassed by another feature that will be included in a forthcoming work. For instance, commands like `[[def:python:... ]]`, `[[def:cpp:... ]]` and `[[def:java:... ]]` will make MCTest generate the corresponding PDF-file according to each specified language.

References

[1] S. Pathak and P. Brusilovsky, "Assessing student programming knowledge with web-based dynamic parameterized quizzes," Proc. of ED-MEDIA, pages 24–29, 2002.

[2] P. Brusilovsky and S. Sosnovsky, "Engaging students to work with self-assessment questions: A study of two approaches," ACM SIGCSE Bulletin, volume 37, pages 251–255, 2005.

[3] M. Gangur, "Matlab implementation of the automatic generator of the parameterized tasks," June 2018. [Online]. Available: https://goo.gl/wwATNW

[4] A. Basaran, G. Sezer, H. Özcan, H. F. Ugurdag, E. Argali, and O. E. Eker, "Smart question (sq): Tool for generating multiple-choice test questions," Proceedings of the 8th WSEAS International Conference on Education and Educational Technology, EDU'09, pages 173–177, 2009.

[5] "Sobre o MEGUA," June 2018. [Online]. Available: http://cms.ua.pt/megua

[6] P. Oliveira, D. Seabra, and P. Cruz, "Parametrized problem databases in sage," 2014.

[7] "Tutorial do MEGUA," June 2018. [Online]. Available: http://megua.web.ua.pt/tutorial

[8] "*TestMakPro3*," June 2018. [Online]. Available: http://www.image-ination.com/testmakePro3

[9] "AMC - Multiple Choice Questionnaires management with automated marking," June 2018. [Online]. Available: http://auto-multiple-choice.net

[10] H. Kagotani, F. Bral, A. Bienvene, and A. Sarkar, "Auto Multiple Choice," June 2018. [Online]. Available: http://download.gna.org/auto-qcm/auto-multiple-choice.en.pdf

[11] M. Gusev, S. Ristov, and G. Armenski, "Technologies for interactive learning and assessment content development," *International Journal of Distance Education Technologies (IJDET)*, vol. 14, no. 1, pp. 22–43, 2016.

[12] M. J. Gierl, H. Lai, and S. R. Turner, "Using automatic item generation to create multiple-choice test items," *Medical education*, vol. 46, no. 8, pp. 757–765, 2012.