MakeTests: Generate and Correct Individualized Questions with Many Styles

Fernando Teubl[a], Valério Ramos Batista[b] and Francisco de Assis Zampirolli*[c]
Centro de Matemática, Computação e Cognição, Universidade Federal do ABC (UFABC), 09210-580, Santo André, SP, Brazil

Keywords: Automatic Item Generation, Multiple-choice Questions, Parametrized Quizzes.

Abstract: Hardcopy exams are one of the most traditional evaluation methods. However, the greater the class the more endeavour to correct them. Automatic generation and evaluation tools can help but most of them are restricted to multiple-choice tests. Our paper presents MakeTests, a free of charge and open source system that automates both generation and correction of exams. MakeTests' main contributions are: (1) highly parametrized questions drawn from a database; (2) inclusion of many question styles besides multiple-choice (true/false, matching, numerical and written response, only this one with manual correction); (3) real-time correction with webcam upon handing in. This paper shows how to elaborate exams with MakeTests, specially regarding the several question styles with parametrization, and also how to correct them automatically. These procedures are illustrated here by two experiments: the first one focused on feedback immediately upon handing in a test that was given to 78 students, and the second one managed by a professor without any programming knowledge. Our results indicate that MakeTests allows for quick elaboration of parametric questions and fast correction, even for users that lack technical knowledge.

1 INTRODUCTION

An important and increasing activity is the task of evaluating students with several computational resources, or Information and Communication Technologies (ICT).

ICT can evolve up to activities in virtual laboratories. In (Burkett and Smith, 2016) the authors present a systematic revision that explores the differences between virtual and hands-on laboratories, both in secondary and in undergraduate education. They conclude that the virtual laboratories can complement activities carried out in hands-on laboratories. Another systematic revision is given in (Hakami et al., 2016), which highlights some works devoted to the students' anxiety when they are evaluated through a computer, namely Computer-Based Assessment (CBA) or e-assessment. Here the whole evaluation is performed on the computer, including the student's mark. This suggests carrying out new studies in order to understand the problems related to the acceptance and to the implementation of CBA.

In (Adkins and Linville, 2017) the authors study if more exams change the students' final grades for an introductory programming course in Computer Science. The Spring class had a total of five exams, whereas the Fall class had only three. Statistically speaking, their study finds no difference but the students prefer more exams in order to reduce anxiety and increase both their self-confidence and their motivation to learn. Moreover, the authors cite several works showing that, in general, classes with more evaluations attain a significantly better performance than others'.

According to (Engelhardt et al., 2017), "[a]ssessments are increasingly carried out by means of computers enabling the automatic evaluation of responses, and more efficient (i.e., adaptive) testing". This work presents two experimental approaches to create variations of items/questions, or heterogeneous items. They conducted an empirical study with 983 individuals who were evaluated by ICT, more specifically Heterogeneous Computer-based Assessment Items (HCAI). The participants were 14 to 16 years old and belonged to 34 schools

[a] https://orcid.org/0000-0002-2668-5568
[b] https://orcid.org/0000-0002-8761-2450
[c] https://orcid.org/0000-0002-7707-1793
*Grant: #2018/23561–1, São Paulo Research Foundation (FAPESP).

Teubl, F., Batista, V. and Zampirolli, F.
MakeTests: Generate and Correct Individualized Questions with Many Styles.
DOI: 10.5220/0010337902450253
In *Proceedings of the 13th International Conference on Computer Supported Education (CSEDU 2021) - Volume 1*, pages 245-253
ISBN: 978-989-758-502-9

in Germany. This work also discusses the complexity of creating heterogeneous items. For instance, an item can become difficult to understand by changing a single letter in a word or the word order on that item. For evaluations they selected 40 out of 70 items, of which 10 and 30 had hard and easy levels of difficulty, respectively.

A study presented in (Nguyen et al., 2017) was conducted by comparing CBA with traditional methods on 74 undergraduate modules and their 72,377 students. The modules belonged to a variety of disciplines (25% in Science & Technology, 22% in Arts & Social Sciences, 14% in Business & Law, 9% in Education & Languages, and 30% in others). The authors found that the time devoted to evaluation activities had a significant relation to the passing rates. Their work also concluded that the balance between weekly evaluations and other activities by CBA has a positive influence on the passing rates.

Therefore, if we have ICT that make teachers' tasks easier when they conduct more evaluations (Adkins and Linville, 2017; Nguyen et al., 2017; Nguyen et al., 2018) with heterogeneous items (Engelhardt et al., 2017) and paper-and-pencil (as verified in (Hakami et al., 2016)), but generating and correcting tests automatically, the students will probably achieve a better performance.

In this paper we present the system MakeTests, whose open source code is available on GitHub. Its code can be easily adapted to new question types, useful for the generation and correction of printed exams. Section 2 motivates the discussion of some related works compared with MakeTests. In Section 3 we explain how to use MakeTests' method, later exemplified by two experiments described in Section 4. Finally, some future work and conclusions are drawn in Section 5.

2 RELATED WORKS

The Introduction presented studies that reveal the importance of making frequent assessments. Now we comment on some works more related to what is proposed in this article, namely ICT that facilitate the process of creating and correcting heterogeneous (or parametric) questions of various styles.

In (Smirnov and Bogun, 2011) the authors present an ICT resource of visual modelling to teach science and mathematics that includes solving scientific problems. Implemented with PHP programming language and MySQL databases, the ICT relies on uniform relational database of teachers, students, educational projects and educational studies. The authors tested the ICT for over 1,000 students of secondary schools in Russia. As an example, groups of 5-6 students had to solve problems with Newton's Second Law. They filled out tables of values, visualized graphics and tried to decide on the problem analytically. However, the authors did not work on actual graphical interfaces but focused on the methods to create activities. Therefore we cannot draw conclusions about the usability of their ICT regarding students' performance in tests.

The formal specification language ADLES was introduced in (de Leon et al., 2018; Allen et al., 2019). It is open-source and devoted to formal specification of hands-on exercises about virtual computing, networking, and cybersecurity. With ADLES educators can design, specify, and semi-automatically deploy a virtual machine (VM) for classes, tutorials or competitions. Students access the VM in order to accomplish tasks.

In (Zampirolli et al., 2019) the authors present the MCTest platform vision.ufabc.edu.br, which is developed in Django and MySQL, whose open source code is available on GitHub. Hence one can install this platform in several institutions, and the system administrator (SA) will register the departments, courses, disciplines and professors. To each course the coordinators attribute Topics, Classes, Questions, Exams, Professors and Students. Any professor can also create Classes, Questions and Exams. All these entities are created on web browser windows. Classes and Questions can also be imported from CSV files, in which the students' Id, name and email are specified in the case of a Class, but Questions follow another CSV formatting. The purpose of that paper is to describe the process of creating parametric questions of either dissertation or multiple-choice type which rely on some Python code in their scope. In this way, MCTest produces individualized exams, one for each student, but they are all contained in a single PDF file for each class. Moreover, a professor that lectures a course can generate a unified exam for all of their classes. The correction is automatic for multiple-choice questions providing the professor digitizes the answer cards into another PDF to be uploaded by the system. For questions that include program codes the student can submit the answers to Moodle (moodle.org) via VPL (available at vpl.dis.ulpgc.es) for automatic correction. As we are going to see in the next sections, MakeTests includes more different question styles than MCTest.

With the intention of reducing plagiarism (Manoharan, 2019) reports how significant it is to create personalized multiple-choice questions. That work describes the insufficiency of just shuffling questions and their respective order of alternatives. One must also elaborate efficient *distractors*, namely the wrong alternatives, otherwise students can simply discard them to guess the right answer.

According to (Manoharan, 2019) there are normally three approaches of personalized evaluations: 1) parametrization, in which some parameters take random values; 2) databank, from which questions are selected at random; 3) macro, which is a program fragment (inside a question) that is replaced by a new phrase whenever executed. Their multiple-choice tests are responded on *optical answer sheet*, also called *bubble/Scantron sheet*, commercialized by Scantron Corporation®. On this sheet the student must write both the script Id and theirs, because together they indicate what exam variation was answered by that student.

The differences between MakeTests and these related works are that with MakeTests the professor develops questions by means of various Python scientific libraries such as graphics (`matplotlib`), algebraic analysis (`sympy`), and many others. Activities are individualized and generated in hard copy, and they make use of various styles of questions. Corrections can be carried out automatically through a mobile device. The professor can also resort to answer keys generated by MakeTests in order to facilitate corrections, as we are going to discuss in the next sections.

MakeTests is ideal for examiners that prefer written response tests because, in the case of a medium difficulty test, the automatic correction will already show the student's right and wrong answers. Hence the examiner can concentrate their attention only on what avails of the student's solutions that led to wrong answers. Roughly speaking, the examiner's manual correction will be just half of the whole work. Of course, in the case of large classes MCTest can spare even the whole manual correction, useful when the examiner just wants to give a preparatory exam.

MakeTests' greatest advantage is that it enables configuration and elaboration of exams in a highly flexible way. MakeTests profits the user that is familiar with Python language, shell commands and JSON format. Otherwise one can resort to templates that are easy to adapt, since question elaboration is what mostly needs teachers' and professors' endeavour.

Each question prepared with MakeTests is represented by an abstract Class (in OOP) that defines a model, namely the user must implement it according to its aims. The question consists of a text, which may include parametrization, and the answer format, which defines its type and also its procedure of correction. Thus MakeTests enables an unlimited diversity of questions and answer types, which just rely on their implementation. The next section explains the details of our methodology.

3 METHOD

Here we describe all the necessary steps to elaborate and generate exams through MakeTests. The first one consists of creating a Database of Questions (DQ) and classifying them under groups based on subject, difficulty or any other characteristic, as detailed in Subsection 3.1. An example of a typical question in MakeTests will be given in Section 3.2. In the second step exams are generated according to a configuration set by the user, as explained in Subsection 3.3. This configuration encompasses the layout of the exam (header, fonts, logotype, etc.) and also the groups of questions that will be drawn with their respective weights. Here one must give the DQ path, as well as the one to the class roll. After generating the tests the user can print them for the examiner. Finally Subsection 3.4 presents the third step, namely the automatic correction, of which MakeTests allows for three types:

Manual. The user prints an answer key from MakeTests in order to correct the exams manually;

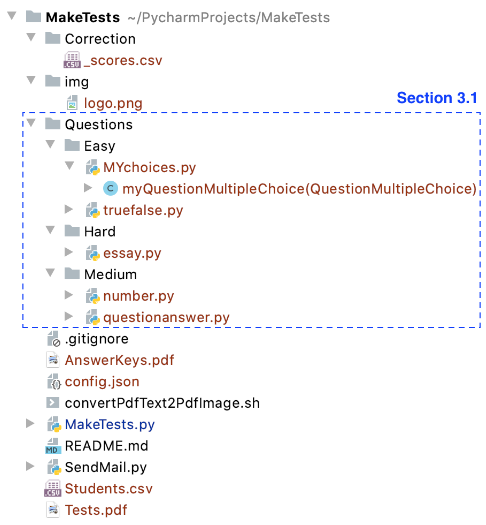
Scanned. The user scans all the exams into a single PDF that MakeTests will process and render a spreadsheet with marks and feedback;

Real-time. The user corrects the exams instantly with a (web)cam by pointing it to the answer sheet. This also generates the aforementioned spreadsheet.

The spreadsheet comes in CSV-format and the feedback lists each question number with the respective student's answer and score. Complementarily, the library `SendMail.py` was developed to dispatch this student's information to their private email address, so that each student will promptly get an individual summary of their performance.

All these resources are organized in a structure of files and folders generated by `MakeTests.py`, a library that is available on github.com/fernandoteubl/MakeTests. Figure 1 illustrates the directory structure, and besides `MakeTests.py` (only 2,609 lines including examples and comments) the reader will also find another three files on GitHub: `SendMail.py` (only 270 lines), `README.md` and `convertPdfText2PdfImage.sh`. All MakeTests code was developed in Python 3.8.

The next subsections will explain each of the aforementioned steps.



```
▼ MakeTests ~/PycharmProjects/MakeTests
  ▼ Correction
      _scores.csv
  ▼ img
      logo.png                                          Section 3.1
  ▼ Questions
    ▼ Easy
      ▼ MYchoices.py
        ▶ myQuestionMultipleChoice(QuestionMultipleChoice)
      ▶ truefalse.py
    ▼ Hard
      ▶ essay.py
    ▼ Medium
      ▶ number.py
      ▶ questionanswer.py
    .gitignore
    AnswerKeys.pdf
    config.json
    convertPdfText2PdfImage.sh
  ▶ MakeTests.py
    README.md
  ▶ SendMail.py
    Students.csv
    Tests.pdf
```

Figure 1: File and folder structure of MakeTests. The master file is `MakeTests.py`, which can generate template questions indicated by the dotted rectangle. It can also produce a student list for an exam (`Students.csv`), all the exam issues in a single file (`Tests.pdf`), answer key (`AnswerKeys.pdf`), and so on. These resources and their employment are all detailed in `README.md`.

3.1 Elaborating Questions

The DQ is classified according to groups represented by directories identified with a dotted rectangle in Figure 1. For a large DQ one can also add subdirectories that indicate subgroups, which make it easier to organize the whole set of questions. For instance, questions can be classified by theme in directories, together with their respective level of difficulty in subdirectories. A question of an exam issue corresponds to a unique Python file, and reciprocally. For example, suppose the user runs MakeTests for an exam to have three questions comprising all the difficulty levels in Figure 1. Then for each exam issue MakeTests draws one of the files in the folder `Medium`. An error is reported if a folder lacks the minimum number for the required specification.

Each question is represented by a Python class implemented in such a way that it contains all the necessary information to generate the *object question*, from which MakeTests will render its final layout. Each object question acquires specific data of a student contained in the class roll `Students.csv`, like name, Id and email, this file depicted in Figure 1. As a Python Class, the user can parametrize any of its parts. MakeTests furnishes pre-implemented codes that simplify the elaboration of new questions whenever the user creates new corresponding PY-files. In the present day MakeTests comprises the following Classes for question styles:

Essay. Written response to be corrected manually, and the score recorded in a spreadsheet of students' marks;

MultipleChoices. An Array Class, in which any element is a typical multiple-choice question. The user defines the number of questions to be drawn;

TrueOrFalse. An Array Class in which any element is a group of questions, and the answer key of any group is a Boolean sequence. The user defines groups with their respective questions, and afterwards the number of groups to be drawn;

QuestionAnswer (QA). A Class that generates matching questions. The user defines a list of questions and another of answers;

Number. The student chooses digits in a matrix to form a number. The user defines the expected number of digits and the tolerance (zero for exact and weighted for approximate answer).

3.2 Example

Here we detail an example of parametrized multiple-choice question named `MYchoices.py` in Figure 1. In order to see its layout the user can run the following on the shell:

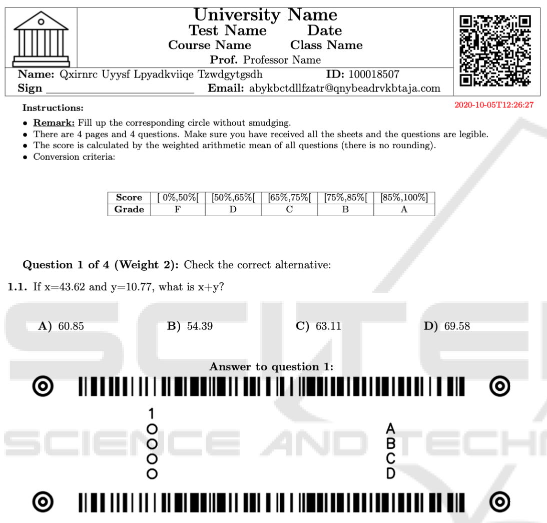
```
./MakeTests.py -e choices >
                   Questions/Easy/Mychoices.py
```

In `MYchoices.py` we implemented a Class named `myQuestionMultipleChoice`, whose code is reproduced in Figure 2. There one sees the method `makeSetup`, which creates two floating-point variables `x` and `y` on lines 10-11. These take random values between 1 and 49 with two decimals. On line 6 we see `random.choice(['+'])` taking only the addition operation. In order to enable all basic arithmetic operations we can replace this line with `random.choice(['+','-','*','/','**'])`. Moreover, lines 12-16 must be encapsulated by the conditional `if op == '+':` and adapted for the other operations. In fact MakeTests will shuffle the entries of `vetAnswers` for each exam issue, no matter what operation was drawn to define this vector. On lines 12-15 notice that `round` is superfluous for `'+'` and `'-'`, but necessary for the others. Hence we kept it because `MYchoices.py` can serve as a template that will require little customization from future users. By the way, this is one of the mindsets that enable good employment of MakeTests.

With MakeTests one can utilize, modify or even create question styles, which in their turn will allow for many question types. As an example, the aforementioned class QA for the matching style enables us to work with the types one-to-one, one-to-many and

```python
from MakeTests import QuestionMultipleChoice
class myQuestionMultipleChoice(QuestionMultipleChoice):
    def makeSetup(self):
        import random
        self.questionDescription = "Check the correct alternative:"
        self.questions = [self.mySubQuestion(random.choice(['+']))]
        self.correctionCriteriaDescription = ""
    def mySubQuestion(self, op='+'):
        import random
        x = round(random.uniform(1, 49), 2)
        y = round(random.uniform(1, 49), 2)
        vetAnswers = [[round(x + y, 2), True],
                      [round(x + 1.2 * y, 2), False],
                      [round(1.2 * x + y, 2), False],
                      [round(1.2 * x + 1.2 * y, 2), False]
                      ]
        return {"statement": '''If x={x} and y={y}, what is
            x{op}y?'''.format(x=x, y=y, op=op),
                "alternatives": sorted(vetAnswers, key=lambda k:
                    random.random()), "itensPerRow": 4}
    def calculateScore(self, correct, wrong, blank):
        if   correct == 1: return 100
        else:              return   0
```

Figure 2: Code of a question implemented in the Class myQuestionMultipleChoice.



University Name
Test Name   Date
Course Name   Class Name
Prof. Professor Name

Name: Qxirnrc Uyysf Lpyadkviiqe Tzwdgytgsdh   **ID:** 100018507
Sign ____________________   **Email:** abykbctdllfzatr@qnybeadrvkbtaja.com

2020-10-05T12:26:27

Instructions:

- **Remark:** Fill up the corresponding circle without smudging.
- There are 4 pages and 4 questions. Make sure you have received all the sheets and the questions are legible.
- The score is calculated by the weighted arithmetic mean of all questions (there is no rounding).
- Conversion criteria:

Score	[0%,50%[	[50%,65%[	[65%,75%[	[75%,85%[	[85%,100%]
Grade	F	D	C	B	A

Question 1 of 4 (Weight 2): Check the correct alternative:

1.1. If x=43.62 and y=10.77, what is x+y?

A) 60.85   **B)** 54.39   **C)** 63.11   **D)** 69.58

Answer to question 1:

Figure 3: Cutout of an exam that includes the question in Figure 2. Here we see the header and the answer card for the automatic correction. A single answer card can be configured to comprise many such questions at once.

many-to-many. Hence, for each style the user can aggregate new types, either by resorting to an already implemented Class or by creating a new one from scratch.

Parametrization allows the question in Figure 2 for many possible renderizations, one of them depicted in Figure 3, which will be discussed in the next subsection. For instance, in order to create an exam by drawing eight random questions from '+', '-', '*', '/', '**', we can simply adapt the code in Figure 2 with the function `random.choices`, as mentioned before. Moreover, the new questions must be added on a list, so that Figure 3 will have eight columns, one for each question, and the ninth column will remain with the letters A, B, C and D, as depicted there.

3.3 Configuring the Exam

All the information regarding the generation of an exam to a specific class must be configured through a JSON file. MakeTests outputs a standard JSON file that can be changed by the user. For instance, in order to get the file `config.json` in Figure 1 we run the shell command:

```
./MakeTests.py -e config > config.json
```

The JSON format was adopted in MakeTests because of its both simple representation and easy portability to web systems. As a matter of fact, MakeTests handles an *extended* JSON syntax that includes comments and line breaks. In order to generate the exam in PDF from JSON this file is endowed with fundamental fields handled by MakeTests, namely:

IncludeJSON. Allows for merging another JSON file to modify or complement the original one;

Questions. Contains both the DQ path and the configuration to select questions, as depicted in Figure 4, which shows the parameters `db_path` and `select`, respectively. Each question of an exam issue is drawn from a group, and this field also defines the weight of the corresponding question, together with a prefix as exemplified in Figure 4;

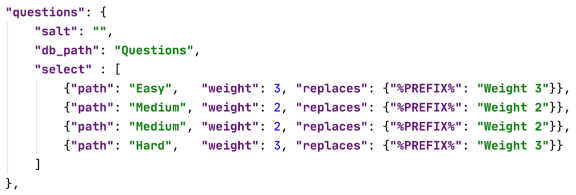
Input. Contains the class roll in CSV. For instance, in order to produce the header in Figure 3 we used the default file `Students.csv` generated by MakeTests;

Output. Sets path and file names of both the PDF that contains all exam issues and the PDF with the corresponding answer keys. MakeTests suggests creating the path in case it does not exist (for **questions** and **input** the user just gets an error message in this case). In Figure 1 they appear as `Tests.pdf` and `AnswerKeys.pdf`, respectively;

Correction. Sets all the criteria for the automatic correction. In this field we write a Python code to compute the total score, choose a name for the CSV file with the students' scores, and also the directory to store this file and the image of the corrected exams.

Tex. Contains all the information about the exam formatting, like LaTeX preamble, headers and structuring of questions. Notice that MakeTests neither contains nor produces anything in LaTeX, but JSON instead (e.g. `config.json` in Figure 1). Therefore, we can alter the JSON file to generate the exams in HTML.

We have chosen to configure exams in JSON because this gives the user total flexibility to design them. The users can create their own models with customized images and headers, and so relegate the JSON file to specific details like date of the exam, class roll and DQ.



```
"questions": {
    "salt": "",
    "db_path": "Questions",
    "select" : [
        {"path": "Easy",   "weight": 3, "replaces": {"%PREFIX%": "Weight 3"}},
        {"path": "Medium", "weight": 2, "replaces": {"%PREFIX%": "Weight 2"}},
        {"path": "Medium", "weight": 2, "replaces": {"%PREFIX%": "Weight 2"}},
        {"path": "Hard",   "weight": 3, "replaces": {"%PREFIX%": "Weight 3"}}
    ]
},
```

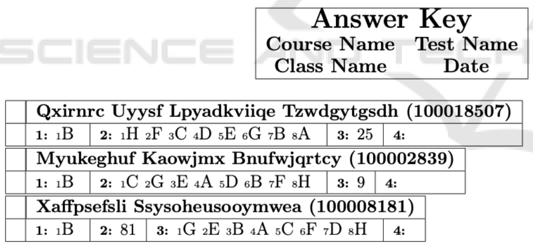
Figure 4: Cutout of a typical `config.json` to create an exam with four questions (one easy, two medium and one difficult with their respective weights). Each path must be as depicted in Figure 1.

3.4 Correcting the Exam

There are three means to correct exams with MakeTests, as explained below.

3.4.1 Manual Correction

The user can print the aforementioned file `AnswerKeys.pdf` as a guide to speed up the manual correction, and also do it in parallel with Teaching Assistants (TAs) without a computer. They can take notes on the hardcopy whenever they find mistakes in the answer keys, which might happen in case a question statement gives rise to an interpretation other than that made by the professor. Figure 5 shows part of such a file that begins with the student "Qxirnrc" from Figure 3.



Answer Key
Course Name **Test Name**
Class Name **Date**

	Qxirnrc Uyysf Lpyadkviiqe Tzwdgytgsdh (100018507)			
	1: $_1$B	**2:** $_1$H $_2$F $_3$C $_4$D $_5$E $_6$G $_7$B $_8$A	**3:** 25	**4:**
	Myukeghuf Kaowjmx Bnufwjqrtcy (100002839)			
	1: $_1$B	**2:** $_1$C $_2$G $_3$E $_4$A $_5$D $_6$B $_7$F $_8$H	**3:** 9	**4:**
	Xaffpsefsli Ssysoheusooymwea (100008181)			
	1: $_1$B	**2:** 81	**3:** $_1$G $_2$E $_3$B $_4$A $_5$C $_6$F $_7$D $_8$H	**4:**

Figure 5: Cutout of `AnswerKeys.pdf` generated by MakeTests.

The two other means of correction resort to Computer Vision, and they generate automatic reports with the students' scores and also feedback. As a matter of fact, they complement the manual correction in the case of written response tests. For instance, the automatic correction shows the student's right and wrong answers and therefore the manual correctors may restrict their work on what avails of the student's solutions that led to wrong answers. Roughly speaking, their whole work will be halved for a medium difficulty test.

Anyway, we shall see that the automatic corrections allow for emailing a feedback to each student even in real time, namely at the moment they hand the test in. Afterwards the professor can digitize the corrected exams and email them to the students, so that each one will receive only their corresponding corrected test.

In the case of a mere preparatory exam just to evaluate the students' performance, the professor can opt for a traditional multiple-choice test and profit MakeTests' DQ. In this case each student can get their scores immediately upon handing in the exam.

3.4.2 Digitizing the Solved Exams

This consists of piling up the solved exams and scanning them all at once through a document feeder. The user gets a single PDF file, or even separate ones at will, to be processed by MakeTests. For the automatic correction MakeTests needs to access the DQ, the class roll and the exam configuration. Each student's name, Id and scores are saved as an individual row of a CSV-file that the professor can open with a spreadsheet program. For each student MakeTests also creates a folder containing their scanned exam with the corresponding answer key, and also the image of each solved question. With MakeTests the professor can get these folders separately compressed and sent to the respective students. Hence, in case of a distance learning course the professor does not have to schedule a meeting with the whole class to discuss the answer key. Therefore, any student who takes objection can furnish arguments with the separate image(s) through email.

Figure 6 shows some items of such a student's folder with the five question styles already available in MakeTests. These images were generated with the shell command:

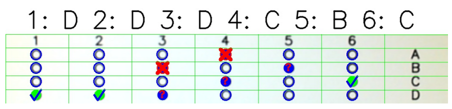
```
./MakeTests.py -p scannerFile.pdf
```

Notice in Figure 6 that a red cross indicates a wrong choice, whereas the missed right one is shown with a red question mark in a blue background. Right answers are indicated by a checked green circle. Exception is made for Figure 6(e), in which MakeTests promptly chooses the computed mark. This one can be changed by the professor in order to include the correction of written response questions.

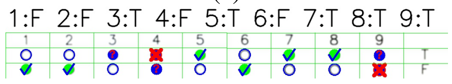
The individual corrections are emailed to the students by the `SendMail` tool available in MakeTests.

3.4.3 Real-time Correction

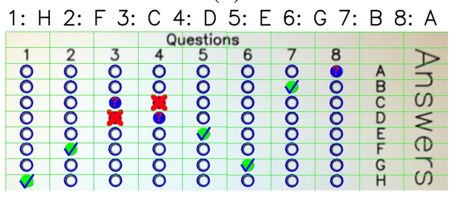
The real-time correction is similar to the one explained in Sub-subsection 3.4.2. MakeTests uses Computer Vision to read the answer cards of Figure 6 and correct them. Student information is contained in the barcodes, as depicted in Figure 3. The student's scores are automatically computed and then sent as feedback to their personal email.



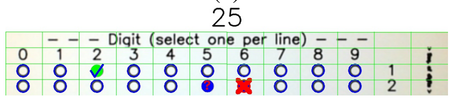
(a)



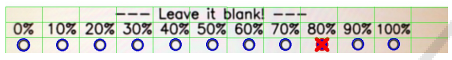
(b)



(c)



(d)



(e)

Figure 6: Cutouts generated by MakeTests after submitting the PDF to the automatic correction: (a) Multiple-Choice; (b) TrueOrFalse; (c) QA; (d) Number (they got a wrong answer 26 instead of 25); (e) Essay.

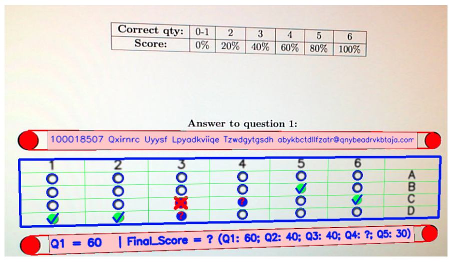
This process is enabled by choosing either the built-in camera of the professor's computer, or a USB webcam attached thereto, namely with the shell command:

```
./MakeTests.py -w 0
```

where 0 stands for the former, to be replaced with 1 for the latter.

With the real-time correction each student gets a lower-bound estimate of their scores already at the time they hand in the exam. Of course, manual correction will later add whatever avails of the handwritten solutions, and also the scores of the dissertation questions. Figure 7 exemplifies a webcam image of an answer card processed by MakeTests.

Although the feedback will be generated again with the second correction method, discussed in Subsubsection 3.4.2, some students might want the real-time correction in order to organize their time for upcoming main exams. But a conventional webcam may generate images in poor quality, and therefore the second method will fine-tune the feedback sent to the students.



Correct qty:	0-1	2	3	4	5	6
Score:	0%	20%	40%	60%	80%	100%

Answer to question 1:

100018507 Qxirnrc Uyysf Lpyadkviiqe Tzwdgytgsdh abykbctdlfzatr@qnybeadrvkbtaja.com

Q1 = 60 | Final_Score = ? (Q1: 60; Q2: 40; Q3: 40; Q4: ?; Q5: 30)

Figure 7: Cutout of an image captured by webcam. Here we see an exam whose first question consists of six parts (1.1 to 1.6), and each one is a multiple-choice item with four alternatives (A to D). Part 1.4 was left blank but MakeTests indicates any missed right answer by a blue circle with a red question mark.

4 RESULTS AND DISCUSSIONS

Now we present some preliminary results obtained with MakeTests which aim at assessing its potential use in new evaluation modalities. All the practical exams involving MakeTests happened at our institution for different courses of the programme Bachelor in Science and Technology (BST).

4.1 First Experiment Report

We tested our third correction method in a class with 78 students of the course Object-Oriented Programming (OOP). The main exam was generated by MakeTests and consisted of three parametric questions, namely:

Q1) Multiple-choice. The statement shows part of a parametrized source code for the student to deduce what it does, and only one alternative describes it correctly. Four parts of source codes were chosen from a DQ at random;

Q2) TrueOrFalse. Some sentences about OOP concepts are listed, and the student must decide which ones are true or false. With MakeTests we selected twelve parametric sentences in a DQ at random;

Q3) QA. The statement includes a two-column table, the first containing several software Design Patterns (DP) and the second one the respective descriptions but in a shuffled order. The student must find which description matches the DP. Five items were randomly selected from a DQ.

Correction was performed according to Subsubsection 3.4.3, and students could promptly check their scores by email. In that experiment we used a USB webcam Microsoft HD 720p 30fps connected to an Apple MacBook Dual Core 512GB SSD with 8GB RAM. As depicted in Figure 7, the examiner just had to point the camera to the answer cards with barely good enframing and ambient light (no flash). MakeTests also sends the scores to the examiner's computer, so that students could also glance at its screen for a quick evaluation. Each exam was corrected in less than 20s, hence the queue was always short, but also because students were previously warned that questions could not be asked at that moment. Moreover, about 20% of the students turned down the automatic correction.

After the exam we proceeded to the method discussed in Sub-subsection 3.4.2. As already explained, this second correction enables MakeTests to process answercards without either distortion or blurring. Moreover, the PDF contains the image of each student's complete exam, which can be emailed to them for accurate checks. Indeed, that was done in the experiment, as described at the beginning of Sub-subsection 3.4.2. See Figure 6 for an example of what the student receives by email. Of course, this feedback is optional and many professors prefer to discuss exam corrections privately with students during the assistance.

On this experiment we remark two important facts:

• The professor detected no cheating in the test. Since questions are parametrized and randomly drawn to each student, then cheating was significantly hampered;

• Far fewer students took objection to their marks compared with standard written response exams already given by that same professor. Though he has not analysed this reduction statistically, he claimed that the usual amount of recorrection requirements dropped more than 50%. This is possibly because that exam did not include any written response question, besides the fact that the students had already received the correction by email.

• The professor detected no cheating in the test. Since questions are parametrized and randomly drawn to each student, then cheating was significantly hampered;

• Far fewer students took objection to their marks compared with standard written response exams already given by that same professor. Though he has not analysed this reduction statistically, he claimed that the usual amount of recorrection requirements dropped more than 50%. This is possibly because that exam did not include any written response question, besides the fact that the students had already received the correction by email.

A difficulty arises in the case of large classes if the professor wants nominal exams, as shown in Figure 3 for the student "Qxirnrc". In the experiment it took almost 10min for the class to begin the test. The professor could have reduced this time by warning the students to look for their seats in alphabetical order.

Another strategy is to replace nominal exams with at least five versions of the same test. Hence, any student that sits beside or behind a colleague will not have more than 20% chance of receiving the same version. Moreover, the professor can print extra copies lest some of the students smudge their answer cards. Such a strategy requires a manual association of each student with their corresponding version but is useful in the case of late matriculated ones that may turn up to sit the exam.

During that exam circa 2% of the answer cards were smudged, so the examiner had to use the answer keys to correct them manually (notice that each issue had three answer cards). Sometimes smudges can be erased by a correction pen/tape/liquid/etc., but Computer Vision cannot always decide which alternative the student really wanted to choose. In this case MakeTests' default is to give naught to the answer, and in Figure 6 we would get a column with an *extra* red cross.

4.2 Second Experiment Report

MakeTests' present day version requires intermediate knowledge in Python programming. However, the professor can draft questions and resort to a technician that will implement them in Python. Here we comment on a maths professor who does not have programming knowledge but was willing to use MakeTests for a test. He gave an extra exam to 36 students of the course Foundations of Mathematics. His draft was implemented by a technician, and consisted of three questions:

Multiple Steps: similar to the one whose answer card is shown in Figure 7 but with five steps 1.1 to 1.5 of a mathematical proof. In one of the versions the statement was "prove that $x > y$ and $y > z$ imply $x > z$". Each step had six alternatives, and only one justified that step correctly.

Multiple-choice with Justification: similar to a standard multiple-choice question but requiring a written response besides choosing an alternative. Naught is promptly given to any wrong choice but the professor additionally checks the student's solution if it led to the right one.

TrueOrFalse: on a list of mathematical proofs the student must find out which are wrong or right.

The technician used the professor's draft to implement the exam with MakeTests. Afterwards the professor gave him all the solved exams digitized in a single PDF, so that MakeTests generated the CSV spreadsheet with each question number and the respective student's answer and score, as mentioned in Section 3. The professor just had the task of validating the second question manually in case of right choice. He expressed a positive opinion regarding the exam variations and the quick correction by MakeTests, but the only hindrance was his lack of programming knowledge.

5 CONCLUSION AND FUTURE WORKS

We have just presented MakeTests, a platform that automates both generation and correction of numerous exam issues via random parameters in Python. Printed exams can be real-time corrected and all the feedback emailed to the students individually. Moreover, MakeTests' high flexibility allows for creating various types and styles of questions. These are MakeTests' main contributions, which to the best of our knowledge also characterizes it as an original work.

Five styles are already implemented in MakeTests, as explained in Section 3.1. For a user to profit MakeTests without anyone else's help, the only requirement is a good knowledge of Python programming. But this can be circumvented by any institution that counts on programming support.

Future versions of MakeTests will work with an interface that exempts users from programming in Python. In this case the user can write an exam in plain text, Markdown or even in LaTeX, and MakeTests will translate it to get the exams in Python.

As another improvement MakeTests will generate online exams not only in PDF, but also in XML compatible with Moodle. Finally, MakeTests' typical DQ is expected to migrate to a Databank of Questions shared by many professors who contribute with their lists of exercises, so that users may simply choose questions to configure their exams instead of elaborating new exercises. Not only MakeTests' DQ but also its whole code could migrate to a WebService, so that professors will be able to keep everything in cloud and synchronized, without needing to access their local machines.

REFERENCES

Adkins, J. K. and Linville, D. (2017). Testing frequency in an introductory computer programming course. *Information Systems Education Journal*, 15(3):22.

Allen, J., de Leon, D. C., and Goes, Haney, M. (2019). Adles v2.0: Managing rapid reconfiguration of complex virtual machine environments. In *The Colloquium for Information System Security Education*.

Burkett, V. C. and Smith, C. (2016). Simulated vs. hands-on laboratory position paper. *Electronic Journal of Science Education*, 20(9):8–24.

de Leon, D. C., Goes, C. E., Haney, M. A., and Krings, A. W. (2018). Adles: Specifying, deploying, and sharing hands-on cyber-exercises. *Computers & Security*, 74:12–40.

Engelhardt, L., Goldhammer, F., Naumann, J., and Frey, A. (2017). Experimental validation strategies for heterogeneous computer-based assessment items. *Computers in Human Behavior*, 76:683–692.

Hakami, Y. A. A., HUSSEI, B., AB RAZAK, C., and ADENUGA, K. I. (2016). Preliminary model for computer based assessment acceptance in developing countries. *Journal of Theoretical & Applied Information Technology*, 85(2).

Manoharan, S. (2019). Cheat-resistant multiple-choice examinations using personalization. *Computers & Education*, 130:139–151.

Nguyen, Q., Rienties, B., Toetenel, L., Ferguson, R., and Whitelock, D. (2017). Examining the designs of computer-based assessment and its impact on student engagement, satisfaction, and pass rates. *Computers in Human Behavior*, 76:703–714.

Nguyen, Q., Thorne, S., and Rienties, B. (2018). How do students engage with computer-based assessments: impact of study breaks on intertemporal engagement and pass rates. *Behaviormetrika*, 45(2):597–614.

Smirnov, E. and Bogun, V. (2011). Science learning with information technologies as a tool for "Scientific Thinking" in engineering education. *Online Submission*.

Zampirolli, F. d. A., Teubl, F., and Batista, V. R. (2019). Online generator and corrector of parametric questions in hard copy useful for the elaboration of thousands of individualized exams. In *11th International Conference on Computer Supported Education*, pages 352–359.