An Automatic Generator and Corrector of Multiple Choice Tests with Random Answer Keys

Francisco de Assis Zampirolli, Valério Ramos Batista, José Artur Quilici-Gonzalez
Federal University of ABC
09210–580 St. André, Brazil
Emails: {fzampirolli, valerio.batista, jose.gonzalez}@ufabc.edu.br

***Abstract*—In the 21st century student-teacher communication in high schools has been increasingly computer-mediated. This brings a strong demand for experienced professionals worldwide, preferably graduated in Information Technology. Until the 19th century the opportunity to attend university courses was a privilege of a few undergraduates, so that the student-teacher relationship was highly personalized. In the 20th century the access to universities increased intensively in many countries, which is a very positive fact. However, it seriously compromised the personal student-teacher communication. This article presents an innovative solution to simplify generation and correction of multiple choice questions (MCQ) in which reliance on the results is wanted. With our program we obtain different issues of the same test with non-coinciding answer keys. Therefore, we offer an original, trustworthy and practical way to evaluate learning in courses attended by a large number of students. Our program generates a LaTeX file and its compiled PDF file containing the individual exam of each student. The exam consists of a front page followed by a list of MCQ and optionally dissertation questions. The front page can be used as an answer sheet of the MCQ, and it has a layout that includes a header. One can opt for individual exams of which the answer keys to the MCQ is unique to each student. All these options are set in a separate configuration file. Our program uses a folder structure to organize classes of students in CSV files and databank of questions in TXT files. The students have to fill out their answer sheets of the front page, which will already contain both the student's name and her/his id number in case of individual tests. After the exam the test front pages of a class can all be scanned into a single PDF file that our software will read to perform the automatic correction. The final scores are stored in a file that contains each student's ID followed by the paired letters given.answer/answer.key of each single question. This file is in CSV format, which is both universally readable and writable by most spreadsheet programs. Our source codes are written in Python programming language. Our program is freely available on the Internet, where users can download the code in Python together with the folder structure and sample files to generate the exams. It has been intensively used at the Federal University of ABC in Brazil, both in classrooms and for distance learning courses, and also in simulations with 6772 tests with excellent results comparable with commercial products. This paper presents a software generator of MCQ which not only issues different versions of the same test, but also performs a fast correction of them all, and exports the results to a CSV file. Thanks to the facilities offered by automatic correction of MCQ the student-teacher communication can concentrate on the essential demands and become feasible again.**

I. Introduction

One of the most striking characteristics of human communication in our 21st century is the computer-mediation. This has already become an inseparable part of the daily life in many branches of society. Regarding Education, in high schools even the student-teacher communication itself now follows the computer-mediated modality in a steadily increasing way [1], [2], [3], [4], [5]. Until the 19th century the opportunity to attend university courses was a privilege of a few undergraduates, so that the student-teacher relationship was highly personalized. In the 20th century the access to universities increased intensively in many countries, which is a very positive fact. However, it seriously compromised the personal student-teacher communication.

But thanks to the automation of some time-consuming tasks, such as correcting a myriad of tests, nowadays teachers are able to devise multiple choice examinations with sufficiently many questions that cover the same topic in different ways [6]. This helps detect individual difficulties. When a student systematically chooses wrong answers to some specific questions she/he can receive additional courseware on the corresponding topic without the teacher's intervention. Clearing up many of these simple doubts by courseware enables the student-teacher communication to concentrate on the essential demands. In this way a personalized student-teacher communication is feasible again.

All over the world, many reputable universities have been offering free distance learning courses. In these courses thousands of students are enrolled simultaneously, including the ones that will be living abroad. Normally all coursewares are meticulously prepared with the assumption that the students will be self-learning. Thanks to the facilities offered by automatic correction of multiple choice tests just a few teaching assistants are necessary to evaluate the students' self-learning process. These assistants will report evaluations to the teacher. Afterwards the teacher can contact some students to help them.

Not only universities have been profiting from this new technology. Also language schools, online learning systems, evaluation systems of secondary schools and others are dealing with knowledge transfer and evaluation in a new way [7].

There are several online learning systems. Here we cite some examples: [8] presents a verification environment of quiz authoring on the internet; [9] presents an improvement of the *iAssing* test package, which is devoted to a popular learning environment called *Moodle* [10]; [11] introduces *TecEval*, an on-line dynamic evaluation system for engineering courses. With this system one can produce questions of four types,

and here we highlight two of them: questions whose answer depends on an equation, and questions whose answer *is* an equation that may include dynamical variables; [12] includes questions that contain program codes for the student to answer what their corresponding outputs are; [13] presents a work in progress that generates gap-fill questions. Firstly, they use machine learning to select the sentences of the test. Secondly, their algorithm chooses a relevant part of each sentence in order to replace it with a gap. Finally, the correct answer is shuffled with effective distractors to make the multiple choices.

Multiple-choice tests need special criteria to score each question adequately. For example, two questions covering the same topic should each count less than a unique question covering another topic [14]. Alternatively, a wrong answer to a question can penalize another one of the same topic [15].

However, automatic correction of test answers on paper is still not properly broadcast. Yet one must resort to the traditional paper tests, because for many purposes online test answers are not reliable. For instance, many language proficiency certifications like TOEFL, ESLAT and DELE still rely on hardcopies to ensure their integrity. Another example is the Brazilian unified evaluation called National High School Exam (abbrev. ENEM in Portuguese), where many attempts of plagiarism have already happened in past occasions.

This article presents an innovative solution to facilitate the generation and the correction of multiple choice tests in which reliance on the results is wanted. With our program we obtain different issues of the same test with non-coinciding answer keys. Therefore, we offer an original, trustworthy and practical way to evaluate learning in courses attended by a large number of students.

II. Method

Our software generates a LaTeX file and its compiled PDF file containing the individual exam of each student. The exam consists of a front page followed by a list of multiple choice questions (MCQ) and optionally dissertation questions. The front page can be used as an answer sheet of the MCQ, and it has a layout that includes a header and a footer. One can opt for individual exams of which the answer keys to the MCQ is unique to each student. All these options are set in a separate configuration file. Our source codes are written in Python 2.7 programming language.

The students have to fill out their answer sheets. In the case of individual tests the answer sheet must be part of the front page, which contains both the student's name and her/his id number. After the exam the test front pages of a class can all be scanned into a single PDF file that our software will read to perform the automatic correction. The final scores are stored in a file that contains each student's identification followed by the paired letters *given.answer/answer.key* of each single question. This file is in CSV format, which is both universally readable and writable by most spreadsheet softwares: LibreOffice Calc, KSpread, Excel, etc. See [16] for details about the CSV format.

Our test generator is called `createTexTests` and its execution is via command line at a terminal prompt. It accesses a plain text file named `config.txt`, of which the header must be changed by the user according to some general explanations that we shall give in the sequel. Please access http://vision.ufabc.edu.br/MCTest4 for a detailed documentation and samples, which are also compressed in the file `MCTest4-master.zip` that you obtain by pressing "Download ZIP" on the right-hand side of the webpage. By extracting this ZIP file you get the folder *MCTest4-master* that we discuss right next.

A. Databanks of Student Classes

The folder *MCTest4-master* contains five subfolders. One of them is called *courses* and is devoted to store distinct classes of either a same course or different ones. As a practical example, suppose you will lecture Object Oriented Programming (OOP) and Software Engineering (SE) in a semester of 2016. If two distinct classes will attend OOP and another three SE then inside *courses* you create two folders *course1* and *course2*, say *OOP* and *SE*, respectively. In the former you will have two lists of student names with their respective ids stored in the files `16_OOP_class1.csv` and `16_OOP_class2.csv`, and in the latter it is just similar. The CSV format will be explained later. For now we must say that generating a test for one of these classes requires you to access line 5 of `config.txt` and write OOP there.

We show these CSV examples in Figure 1, which also summarizes our hierarchy of directories and files. Names in blue indicate that some data will be either created or added in future. Names in green are explained in Section IV.

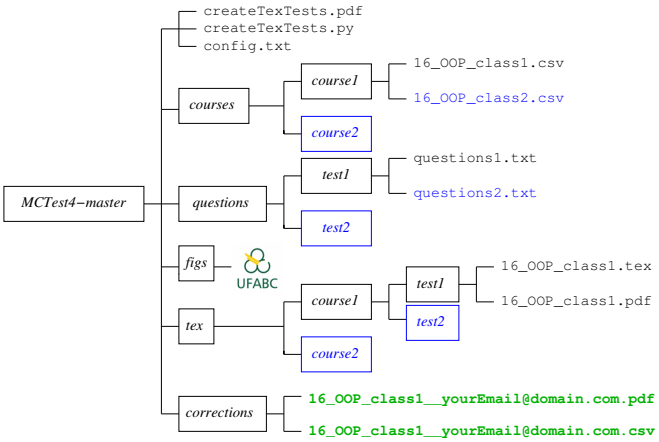


Fig. 1. Directory hierarchy with files.

For instance, when the tests are printed with a different student's name on each issue this means that you have a CSV file with all the students' ids and names. This file must be inside the folder path *MCTest4-master* / *courses* / *course*, where this latter is the namesake of *course1* (or *course2*). Our program `createTexTests` will automatically access this CSV file to include both student's name and id in each header. This file must be named `year_course_class.csv` in order to generate the PDF namesake.

This CSV file has as many lines as students. For example, `12345678;Thing a Ma Bob` as the first line, `23456781;Thing a Ma Jig` as the second, `34567812;What Cha Ma Call It` as the third and so on, where each number is the student's id.

B. Databanks of Questions

Inside *MCTest4-master* we also have the subfolder *questions* for you to store different subfolders. Each of them is designated by a *testname* and contains files that are databanks of questions. Inside *questions* / *testname* you must have at least one plain text file *questions**n***.txt, where ***n*** is an identifier. Again you must access `config.txt` to specify *testname* on its line 6. There are other options in `config.txt` that you can edit. We shall explain them later, but once `config.txt` has been changed accordingly the command line

```
ipython createTexTests.py config.txt
```

will create the subfolder path *course* / *testname* inside the folder *tex*, which is another subfolder of *MCTest4-master*. Inside *testname* we shall have a PDF file with all the exams to be given to that class. This file is named after the corresponding CSV file. In our example it will be `16_OOP_class1.pdf`, and *testname* will also contain both the LaTeX source file and a GAB file. The GAB is in fact a plain text file that contains the answer keys for each student's id. In our example it will be `16_OOP_class1` plus the extra label `__yourEmail@domain.com_GAB` that will be clarified right next. However, in Figure 1 the GAB file is *omitted* for two reasons: Firstly, it should be accessed only by the automatic corrector, not by the user. You can read its content, but this is unnecessary because the automatic corrector will generate a CSV file that lists each student's *given.answer* / *answer.key*, as explained before. Secondly, by accessing the GAB file the user can change it accidentally, and this will compromise fairness when running the automatic corrector.

Figure 1 shows a PDF manual in the folder *MCTest4-master*, which can also be read online at http://vision.ufabc.edu.br/MCTest4. There the user will find instructions to install Python and LaTeX. Maybe you already have an installed LaTeX, but without special packages mentioned in Section IV. In this case the compiler will ask for the missing ones, which can be downloaded from the internet individually. Although `createTexTests` was originally written for large classes of the Brazilian educational system, on line 22 of `config.txt` the specified language is English. Changing it to Portuguese will make `createTexTests` use the LaTeX package "babel" for Portuguese to recognize typed accented letters in that language. If you need to use a third language please contact the authors for help. We can personalize your `createTexTests` and also specify the label `__yourEmail@domain.com_GAB` to your email address. The test corrector will then send the final results to the specified address.

III. Detailing

The folder *MCTest4-master* contains our executable, its documentation, sample files and also five subfolders. As explained before, one of them is *questions* for you to store different subfolders. Each of them is designated by the same *testname* mentioned in Subsection II-A. Inside *questions* / *testname* you must have at least one plain text file *questions**n***.txt, where ***n*** identifies a databank of questions.

Our generator will use all the question files inside *questions* / *testname*, so that you must remove the ones that should not be on the test. In these files each question must start with a determiner. There are three kinds of determiner for multiple choice questions, `QE::`, `QM::`, `QH::`, which stand for the attributed levels of difficulty "easy", "medium" and "high", respectively (this structure was inspired by https://code.google.com/p/criaprova). Optionally you can add the subject of the question and a subclass identifier, which must be either a single letter or digit. This identifier is useful when we do not want the test to have two or more variations of the same question. Here we give an example:

```
QE::Loops::a::The JAVA command{\tt for(int
i=0;i<5;i++) i=5;} will attribute 5 to i
% <- Symbol to white comments if necessary
A: Just once
A: Five times
A: Never (the compiler does not accept it)
A: The answer depends on the JAVA version

QE::Loops::a::The JAVA command{\tt for(int
i=5;i>0;i--) i=5;} will attribute 5 to i
A: Forever (it is an infinite loop)
A: Five times
A: Never (the compiler does not accept it)
A: The answer depends on the JAVA version
```

In our example we give only four choices to answer each question but this number could be five, six, etc. However, it must be the same number for all questions in all files inside *questions* / *testname*. If you choose to generate tests with random answer keys, then the GAB file will always take the first ones in *questions**n***.txt as correct. For instance, one of the outputs the test generator can give for the above example is

1. The JAVA command `for(int i=5; i>0; i--) i=5;` will attribute `5` to `i`
A. The right answer depends on the JAVA version
B. Five times
C. Forever (it is an infinite loop)
D. Never (the compiler does not accept it)

By default, on line 7 of `config.txt` the option for random tests is enabled. If you do not change it, then in our example the GAB file will take C as the correct answer, the first alternative of the corresponding question in the database.

The list of questions is generated after the answer sheet. Figure 2 shows an example with 27 questions and 5 choices per question. We have three columns with 9 questions each, where the middle one was omitted but indicated with $\cdots$, whereas $\vdots$ indicate the same to questions 5-9, 23-27.

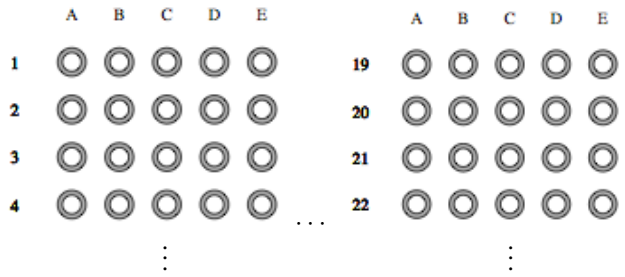


Fig. 2. Example of an answer sheet.

The student is instructed to paint one single white circle per question but without going outside the limits of the gray annulus. This is important because the automatic corrector works with scanned PDF images of the filled out answer sheets. Hence, any excess of stain outside the gray limit will make the corrector loose track of the row of circles to that question. For the student, the automatic corrector will then generate a CSV score file with a false number of questions and/or answers per question.

By editing lines 11 and 12 of `config.txt` one determines the layout of the answer sheet. Figure 2 shows an example in which we chose 9 as the maximal number of questions per column and 4 as the maximum number of columns. But since we had a total of 36 questions then just three columns were printed. A fourth one would appear had we added more questions to the test.

The answer sheet is just part of the front page, which must also include a header. This one contains important data like institution, date, instructions, etc. They are all editable in `config.txt`, but the header also contains non-editable data.

For instance, if you want each student to have a different issue of the test, then our program will access the CSV files mentioned in Subsection II-A to write name and id on the header automatically. But if you just want n different issues for a class with m students, $m >> n$, then your CSV file will only have n lines containing precisely `1;char`$_1$ then `2;char`$_2$ and so on until `n;char`$_n$ as the last line. The characters `char`$_k$, $1 \leq k \leq n$, are for you to quickly identify each issue. They can also be white spaces when n is small enough to recognize each different issue already by the first question.

Optionally, you can add an image to the header. Our program will search the corresponding image file in the subfolder *figs*, as depicted in Figure 1. But be sure that the image already has an adequate dimension for the header, because in `config.txt` you cannot edit its dimensions. Finally, there is another non-editable part of the header created automatically: the barcodes. We are going to discuss them in the next section.

IV. The Automatic Corrector

For the automated decoding of graphical encryptions, scanners have been widely used in various applications such as barcodes and qrcodes. In addition, there are many systems based on Optical Mark Recognition (OMR) that correct multiple choice questions automatically [17], [18]. They use image segmentation techniques that are beyond the scope of our present paper. Therefore, we shall omit details here and leave them to be treated in future works.

As explained in Subsection II-B, our program `createTexTests` generates a GAB file with each student's individual answer keys if you opt for random tests. This is enabled by default on line 7 of `config.txt`.

The corrector reads a PDF file with all scanned test front pages, and then uses the GAB file to generate the scores. The PDF file is exemplified in Figure 1 by `16_OOP_class1__yourEmail@domain.com.pdf`. As explained in Section III the corrector identifies the student's choice for each question by analyzing its row of circles in order to find the marked one. Finding none or two produces *0/answer.key, 2/answer.key* to that question, respectively. A barcode reading error is indicated by ID equals zero, whereas a marking reading error by a wrong number of questions and/or answers per question in the scores. A typical example of this latter happens when a circle is intensely marked outside its limits.

But each front page also comes with a barcode in the header that hides the following information: the student's id or the answer keys to that issue of the test. The corrector deciphers the barcode and then gives the final score after having compared each marked choice with the corresponding answer key of the GAB file. These results are then stored in the file of scores, which in our example of Figure 1 is `16_OOP_class1__yourEmail@domain.com.csv`, as explained in Section II.

The barcodes are generated via LaTeX packages **pstricks** and **pst-barcode**, which follow the Universal Product Code (UPC). If the test supervision is reliable then the students will hardly be able to decipher the barcode with forbidden devices like smartphones. Of course, the hidden answer keys are directly related to the thickness and to the spacing of the bars. But they differ by pixel width, which in practice cannot be recognized with the naked eye.

In spite of that, we can prescribe rules to decipher barcodes differently from the UPC. In this case please contact the authors for help.

Like `createTexTests` the automatic corrector MCTest is freely available for use. However, its source code is *not* open yet. In order to run it the user must send both the GAB and the scanned PDF file to our server by File Transfer Protocol. At the shell prompt you must type

```
ftp vision.ufabc.edu.br
```

then press the enter key, log in with "anonymous" and

void password, then change directory with `cd upload`, `cd MCTest4` and send your files. In our example the two commands at the ftp-prompt are

```
put 16_OOP_class1__yourEmail@domain.com_GAB
put 16_OOP_class1__yourEmail@domain.com.pdf
```

which our program MCTest will detect and process automatically. This program runs every minute and looks for namesake files with extensions GAB and PDF for which the corresponding CSV has not been created yet. There will not be a GAB file if you have disabled the option for random tests, in which case MCTest will run much faster. However, in this case you must have included a test front page with the answer keys as the first one before scanning them all into the PDF file. This is because MCTest will take this first page as the answer key of all others if it does not find the GAB of the corresponding PDF file. Finally, MCTest generates the file of scores, exemplified by `16_OOP_class1__yourEmail@domain.com.csv` in Figure 1. This file will be sent to the specified e-mail, but you can also download it with the following command line:

```
get 16_OOP_class1__yourEmail@domain.com.csv
```

The next section is devoted to discussing some experiments with `createTexTests` and MCTest.

V. Experiments

In this section the user will have an overview of the performance of our softwares. Here we describe three experiments carried out with the automatic generator and corrector of multiple choice tests.

A. First Experiment

The first and second authors lectured the course Object Oriented Programming in 2015. We had evening and morning classes with 66 and 70 students, respectively. Tests were generated from the same questions databank, and both tests had 10 questions with 4 choices each, plus an eleventh dissertation question. The front pages with barcode and answer sheet were all scanned into two distinct PDF files, one for each class. They were both scanned with a resolution of 150 DPI. However, the morning class had only $n = 5$ different issues for $m = 70$, whereas the evening class had a different issue for each student ($n = m = 66$). As explained at the end of Section III, the choice of (n, m) is left to the user.

For the morning class each issue from 1 to 5 had 14, 13, 13, 16 and 14 copies, respectively. The only failure pointed out by our corrector is depicted in Figure 3. It shows an earlier version with circumferences instead of the annuli depicted in Figure 2.

As explained in Section III, any excess of stain outside the boundary will make the corrector generate an atypical CSV score file. In Figure 3 the answer is right but the trace that practically joins 9A with 9B made the corrector count both columns A and B as a single one. This happens because for each scanned page our corrector enframes all images of the answer sheet that appear in the same vertical. But only one test had this problem and it was then corrected separately by hand. Students can use graphite, or else a correction fluid to whiten the stain, but if the circular contour is damaged the corrector can fail again.

The reader will notice that in Figure 3 circumferences 7D and 10A *are* slightly stained outwards. But the corrector tracked them even though, for it identifies the circles within a certain tolerance. In our newest version the annuli have reduced such failures considerably. For the evening class there was absolutely no failure.

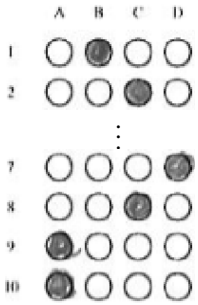


Fig. 3. Failure at filling up circumference 9A on the answer sheet.

B. Second Experiment

This one was applied to a blended learning course that at our university we call *Processing of Information*. It was attended by 130 students who had to turn up only for the written exam. In this case $n = m = 130$, where each exam consisted of one dissertation question and 12 others of multiple choice. The corrector tracked all answers without any failure.

C. Third Experiment

In the Introduction we cited the Brazilian high school unified evaluation ENEM. As a preparatory for this exam we took part in a simulation with 6772 tests to be automatically corrected. Each student sat two exams, one in Mathematics and other in Portuguese. Since it was a simulation, and our program's performance was also compared with the commercial software REMARK, then $n = 1$ and $m = 6772$. We refer the reader to http://remarksoftware.com for details about that software, which however cannot generate random answer keys for multiple choice tests. As mentioned at the end of the Introduction, our program is perhaps the first and still the only one with this characteristic.

Students were either at the end of the Primary School, or in the middle of the Secondary School. The latters had exams with 11 questions in one column, and the formers 13 questions in two columns: 6 on the left and 7 on the right. Figure 4 shows a test of the Secondary School.

This exam took place in schools of 10 different cities simultaneously. Two problems occurred in different schools, but we could re-program our code in order to get round them.

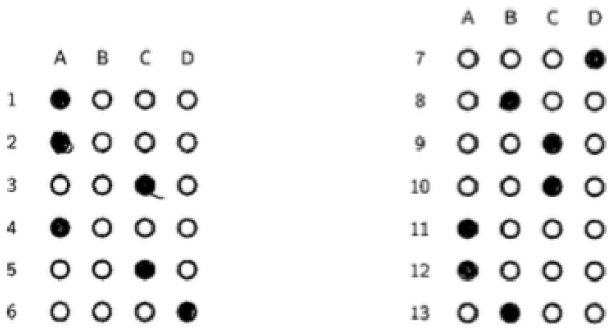


Fig. 4. Answer sheet filled out by one of the students.

All the exams were scanned with only 75 DPI of resolution, so that we had to change the image filters used in our program. Moreover, some of them were printed with low toner, so that the corrector could not read the barcodes properly. Table I shows the output of our corrector to Figure 4.

TABLE I
OUR CORRECTOR'S OUTPUT TO FIGURE 4.

Pag	ID	#Answer	#Question	#Invalid	#Final
53	1234567	4	13	0	7

1	2	3	4	5	6	7	8	9	10	11	12	13
A	A/C	A/C	A/B	C	D	D	B	B	D/C	D/A	A	D/C

Regarding the students at the end of the Primary School, they sat an exam in which Mathematics and Portuguese were both mixed in a total of 11 questions. A comparison between MCTest and REMARK is shown in Table II, in which their numbers date from March 2016 and December 2015, respectively. Our program is under constant improvement and we aim at the professional quality of commercial softwares.

TABLE II
COMPARISON BETWEEN PERFORMANCES OF MCTest AND REMARK.

	Primary School		**Secondary School**	
Tests	3224		3548	
Software	MCTest	REMARK	MCTest	REMARK
Barcode Reading Errors	39	10	63	11
Marking Reading Errors	2	21	0	0
Blank (no Marking)	36	31	17	14
Duplicate Markings	102	65	59	29

We remark that only MCTest has a 100% automatic correction process, because REMARK cannot start correcting before you define where both barcode and answer sheet were placed in the frontpage layout. This is identified by MCTest without any kind of human intervention. Table II shows that our software had many more errors at reading barcodes than REMARK. This happens exactly to the exams printed with low toner, a limitation that future versions will not have any more.

VI. DISCUSSIONS

Among the experiments presented in Section V we choose the 3rd one for a detailed discussion. As explained in Subsection V-C, if the barcode print is faint then MCTest cannot read it properly, unless we change the programmed image filters. However, even with a faint print MCTest reads answer sheets correctly, as exemplified by Figure 5. It shows 3 distinct answer sheets of the test given in the Primary School. The software REMARK was however unable to handle them.

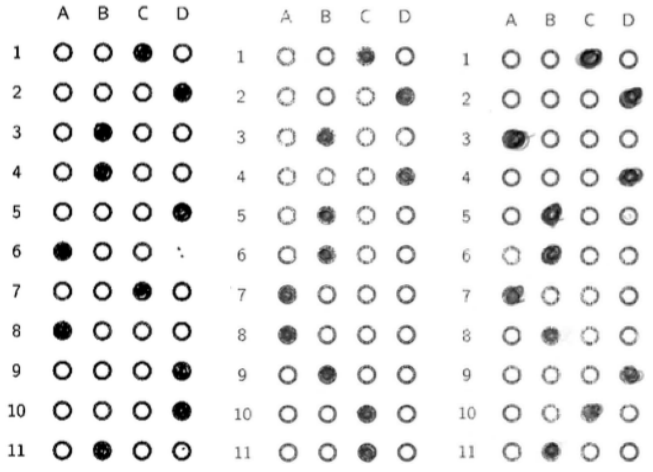


Fig. 5. Answer sheets treated as error by REMARK.

In Figure 5 left, notice that the student practically erased 6D with a correction fluid. This was not a problem to MCTest because we had to re-program the DPI resolution to 75 in that case, as explained in Subsection V-C.

Figure 6 shows an error in the answer sheet that both MCTest and REMARK failed to read due to faint print.

Regarding the 3548 tests in the Secondary School, 67 of them had marking failures that compromised the performance of MCTest and/or REMARK. This number is not the sum of the corresponding four exception cases of Table II for two reasons. Firstly, two or more of them happened on a same scanned front page. Secondly, MCTest and REMARK agreed in 37 cases and disagreed in 30. Anyway, students failed to fill out properly in only 67 / 3548 = 1.88% of the cases, which is a very low rate.

Figure 7 summarizes the distribution of the *Blank* and *Duplicate* cases, 6th and 7th rows of Table II.

Now observe the isolated 4 in Figure 7(b) left. In that case MCTest considered that a total of 17 questions were marked as *Blank*. Of them, 13 happened in common with REMARK. The 4 *Blank* cases occurred on a single answer sheet, depicted in Figure 8. There the faintest markings 6C, 8C, 9D and 10D were not captured by MCTest, while REMARK found them properly.

On the other hand, Figure 9 shows a case in which MCTest found the lightly marked alternative 4B. But there REMARK classified the whole row 4 as *Blank*, which is a curious fact. However, the marking on 6D is so light that both MCTest and REMARK dismissed it and captured just 6B. Again in

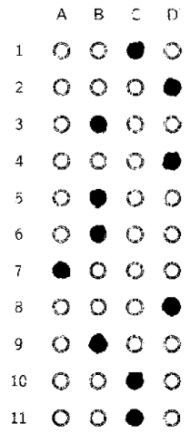
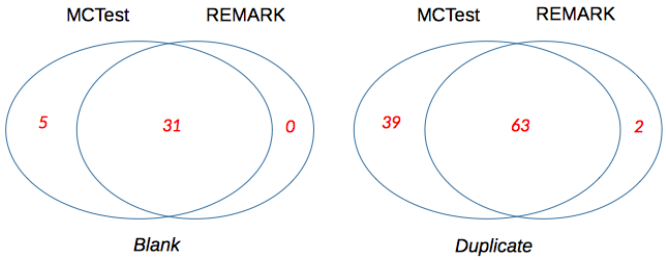
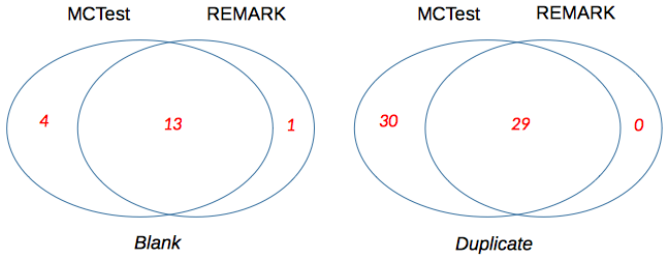


Fig. 6. Answer sheet with marking errors for both MCTest and REMARK.



(a)



(b)

Fig. 7. Distribution of detected *Blank* and *Duplicate* cases of (a) Primary School and (b) Secondary School.

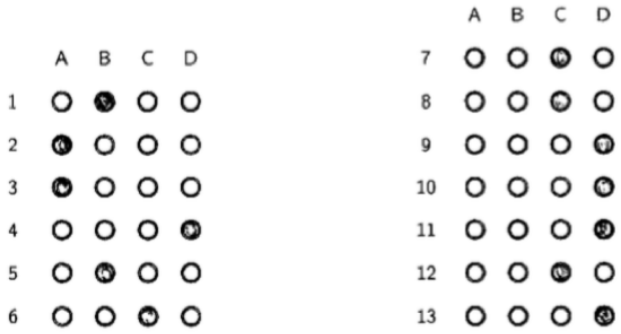


Fig. 8. Answer sheet with too faint markings for MCTest.

Figure 9 one sees that the student marked both 9B and 9C, which was properly identified by MCTest as *Duplicate* (3rd exception of Table II). However, REMARK simply dismissed 9C and considered only 9B.

Fig. 9. A case where MCTest captured more inconsistencies than REMARK.

By the way, circles 9B and 9C of Figure 9 show a typical example that explains why the number of *Duplicate* cases were many more for MCTest than for REMARK. Our image filter is programmed to include circles in a row that were marked with intensities that can differ much more than in the case of REMARK. Personally, we (the authors) do not agree with such procedure of REMARK, which takes for itself the decision about which answer the student chose among two really marked ones. Our MCTest will point out these cases in the file of scores, so that the teachers will be able to take their own decision about them.

VII. Performance

For the 3rd experiment with MCTest we have used microprocessor Intel (R) Core (TM) i7, CPU X980@3.33GHz with 12GB of RAM, operating system Linux Ubuntu 64-bits version 15.04, and executables generated by Python 2.7.9. As explained in Section IV, MCTest runs every minute and looks for namesake files with extension PDF (no GAB files in this case), for which the corresponding CSV has not been created yet. The attributes `nice -20 MCTest.py` include subfolders of */ srv / ftp / upload / MCTest4* in both processes of search and correction. We call it "performance for high priority", which was used in the experiment of Subsection V-C.

As explained in that subsection, the simulation took place in schools of 10 different cities simultaneously. In average we had 8 classes per city, whence a total of 80 PDF files, namely one for each class. The mean number of tests per PDF file was 84.66, and they ranged from 33 to 135. In total our automatic corrector took 102min to process the 6772 tests, thus 0.9s per test.

We could not have access to the performance data of REMARK, hence they are omitted here.

VIII. Conclusion

In this article we presented our solution to a typical problem of our 21st century: the personal student-teacher communication is becoming computer-mediated in a too fast and

increasing way. This is corroborated by a huge demand for new academic computer programs, teaching apps, trained IT personnel, fast networks for distance learning courses, just to mention a few examples. In this context a software that generates and corrects multiple choice tests can be very useful to teachers wishing new tools to save their time, and also for them to help the students more quickly.

After having generated different versions of a multiple choice test with our `createTexTests`, our MCTest can correct the scanned answer sheets in a fast and confidential way. Experimental results with thousands of tests were discussed in Sections V and VI, showing that our tool is very robust and reliable. Both our automatic generator `createTexTests` and automatic corrector MCTest have been developed since 2013. Previous versions did not include a test generator. They only performed an automatic correction. These previous versions were written in the following programming languages: 1) Matlab (correction through processing pictures taken by webcam); 2) Java (for smartphones and tablets with Android); 3) Python (using a PDF file with several tests, similarly to the method presented herein). In all these versions the answer sheets were tables (rectangles instead of circles) drawn in either Microsoft Word or in OpenOffice. Questions and answer keys were neither randomly generated yet. Please see [19], [20], [21] for earlier versions and details.

In its present version the ideal accuracy of 100% was not achieved only because of the problems mentioned in Section VI and Subsection V-C: faint print, low scanning resolution instead of the recommended 150 DPI, and the exception errors listed in Table II. If both print and resolution are of good quality, the institution can decide about accepting or not the tests that contain marking errors. In the second case, students must be warned to check if they received a front page containing an occasional stain either on the barcode or on the answer sheet. If so, they could either receive another one, or clarify it to the supervisors before starting, because for several reasons some institutions do not allow access to the corrected exams.

Future developments include an alternative app in which instead of returning the hardcopy to the teacher, the student scans the answer sheet with a smartphone and sends it directly to a server. The advantage of this alternative is that any inconsistency or problem with the test answers would be promptly detected by the MCTest installed on the smartphone or tablet. The evaluation of the test could be made either by the server or by the smartphone. Future versions of MCTest will include the same technique of [11] to create dynamic questions. In future works we also plan to use an eLearning system such as Moodle. It will enable sharing Databanks of Questions already available for online tests, so that we may also generate and print random exams with these Databanks.

Acknowledgment

We thank the company *Dataeduc - Tecnologia educacional* for conceding the PDF files with the students' examinations, and also for their corrections by REMARK, with which we drew the comparisons presented herein.

References

[1] R. Marín, P. Sanz, O. Coltell, J. Inesta, F. Barber, and D. Corella, "Student-teacher communication directed to computer-based learning environments," *Displays*, vol. 17, no. 3, pp. 167–178, 1997.

[2] M. S. McIsaac, J. M. Blocher, V. Mahes, and C. Vrasidas, "Student and teacher perceptions of interaction in online computer-mediated communication," *Education Media International*, vol. 36, no. 2, pp. 121–131, 1999.

[3] K. Ash, "Teachers make the move to the virtual world," *Education Digest*, vol. 76, no. 5, pp. 32–34, 2011.

[4] R. M. McHugh, C. G. Horner, J. B. Colditz, and T. L. Wallace, "Bridges and barriers adolescent perceptions of student–teacher relationships," *Urban Education*, vol. 48, no. 1, pp. 9–43, 2013.

[5] T. J. Kopcha and C. Alger, "Student teacher communication and performance during a clinical experience supported by a technology-enhanced cognitive apprenticeship," *Computers & Education*, vol. 72, pp. 48–58, 2014.

[6] T. U. Muenchen, "Multiple choice examinations: How to minimize guessing," Retrived from given address, 24 June 2016, last access date. [Online]. Available: https://www.lehren.tum.de/en/topics/examinations/multiple-choice-examinations

[7] S. Bargh, "A model for the implementation of digital examinations: New Zeland's vision for digital assessment," Retrived from given address, 24 June 2016, last access date. [Online]. Available: http://www.iaea.info/documents/paper_226dc25a15.pdf

[8] A. Gordillo, E. Barra, and J. Quemada, "Enhancing web-based learning resources with quizzes through an authoring tool and an audience response system," in *Frontiers in Education Conference (FIE)*. IEEE, 2014, pp. 1–8.

[9] J. R. A. Rodrigues, L. O. Brandao, M. Nascimento, P. Rodrigues, A. A. Brandão, H. Giroire, and O. Auzende, "iquiz: integrated assessment environment to improve moodle quiz," in *Frontiers in Education Conference (FIE)*. IEEE, 2013, pp. 293–295.

[10] M. Dougiamas and P. Taylor, "Moodle: Using learning communities to create an open source course management system," 2003. [Online]. Available: http://moodle.org

[11] J. Jaquez, J. Noguez, G. Aguilar-Sánchez, L. Neri, and A. González-Nucamendi, "TecEval: An on-line dynamic evaluation system for engineering courses available for web browsers and tablets," in *Frontiers in Education Conference (FIE)*. IEEE, 2015, pp. 1–8.

[12] P. Brusilovsky and S. Sosnovsky, "Engaging students to work with self-assessment questions: A study of two approaches," in *ACM SIGCSE Bulletin*, vol. 37, no. 3. ACM, 2005, pp. 251–255.

[13] G. Kumar, R. E. Banchs, and L. F. D'Haro, "Automatic fill-the-blank question generator for student self-assessment," in *Frontiers in Education Conference (FIE)*. IEEE, 2015, pp. 1–3.

[14] J. S. Hsu, T. Leonard, and K.-W. Tsui, "Statistical inference for multiple choice tests," *Psychometrika*, vol. 56, no. 2, pp. 327–348, 1991.

[15] A. Chatterjee, "Better rank assignment in multiple-choice entrance exams," *Current Science*, vol. 105, no. 2, pp. 193–200, 2013.

[16] Y. Shafranovich, "Common format and MIME type for comma-separated values (CSV) files," 2005. [Online]. Available: http://tools.ietf.org/html/rfc4180.html

[17] J. A. Fisteus, A. Pardo, and N. F. García, "Grading multiple choice exams with low-cost and portable computer-vision techniques," *Journal of Science Education and Technology*, vol. 22, no. 4, pp. 560–571, 2013.

[18] T. D. Nguyen, Q. H. Manh, P. B. Minh, L. N. Thanh, and T. M. Hoang, "Efficient and reliable camera based multiple-choice test grading system," in *Advanced Technologies for Communications (ATC), 2011 International Conference on*. IEEE, 2011, pp. 268–271.

[19] F. A. Zampirolli, J. A. Qulici-Gonzalez, and R. Neves, "Automatic correction of multiple-choice tests using digital cameras and image processing," in *Workshop de Visão Computacional, Rio de Janeiro*, 2013.

[20] ——, "Automatic correction of multiple-choice tests on android devices," in *Workshop de Visão Computacional, São Carlos*, 2015, pp. 83–88.

[21] R. T. China, F. A. Zampirolli, R. Neves, and J. A. Qulici-Gonzalez, "An application for automatic multiple-choice test grading on android," *Revista Brasileira de Iniciação Científica*, vol. 3, no. 2, pp. 4–25, 2016.