International Journal of Engineering Education Vol. 31, No. 4, pp. 1058–1070, 2015
Printed in Great Britain

0949-149X/91 $3.00+0.00

Automating the Knowledge Assessment Workflow for Large Student Groups: A Development Experience*

ANDRIJA BOŠNJAKOVIĆ, JELICA PROTIĆ, DRAGAN BOJIĆ and IGOR TARTALJA
Department of Computer Engineering and Informatics, University of Belgrade—School of Electrical Engineering, Bulevar Kralja Aleksandra 73, 11000 Beograd, Serbia. E-mail: andrija@etf.bg.ac.rs, jeca@etf.bg.ac.rs, bojic@etf.bg.ac.rs, tartalja@etf.bg.ac.rs

The paper presents our long-term experience at the University of Belgrade—School of Electrical Engineering in the development and use of the software system called *Test*, designed to automate the knowledge assessment workflow for large student groups in situations that usually require the use of pencil-and-paper testing. The test assembly is done interactively and/or automatically from a database of problems (questions, coding assignments, etc.) and previous tests. In order to enable objective and uniform knowledge tests, a rich set of parameterized problem selection criteria is made available to teachers. Preparation, scanning, and analysis of bubble sheets are highly automated and designed to work with plain paper and regular scanners. The *Test* system integrates with relevant learning management systems and the school's information system. Quantitative measurements have indicated improvements in both test quality of individual tests and uniformity of test quality across multiple tests, compared to the tests assembled manually according to intuitively applied criteria. The system is used for the introductory programming courses at the School. It has been developed through a number of diploma theses. Overall experience showed that the designed system provides an appropriate automation tool, a useful educational platform, and a valuable base for further research.

Keywords: automatic optical inspection; automated test assembly; educational technology; heuristic algorithms; knowledge assessment tool; test authoring tool

1. Introduction

Testing of large student groups often presents an infrastructural challenge for the school administering the tests. For some schools, testing all the students simultaneously is only possible using a conventional pencil-and-paper method. If such a school has multiple exam terms per school year, offers all exams in every term, and has hundreds of students enrolling every year, the time and effort required to support the workflow is immense. Due to the state regulations on higher education and large number of enrolled students, this is the exact situation faced by the ETF (a commonly used acronym for the University of Belgrade—School of Electrical Engineering, Serbia). Although there are numerous off-the-shelf solutions that provide good support for one or more phases of the assessment workflow, we were not able to find an ideal one for use in the specific ETF environment. These conditions motivated the development of a specialized software system called *Test* [1–3].

The *Test* system is primarily intended to support teachers in organizing the database of problems, knowledge tests, and criteria for problem selection, as well as in preparing, scoring, and processing the results of tests, while relying only on common PCs, regular scanners, and plain paper. Instead of the usual terms ''item'' and ''question'', we are using ''problem'' to emphasize the nature of both item types that we use: questions that require analysis and tasks that require synthesis of the solution. Similarly, instead of the usual term ''item bank'', we are using ''database'' due to heterogeneous nature of its contents (problems, tests, criteria, etc.). Each system component is designed to support a certain phase of the assessment workflow: creation of a knowledge test, preparation of corresponding bubble sheets, analysis and scoring of completed bubble sheets, and processing of gathered results. The most relevant features of the system include manual or automatic test assembly, shuffling of questions and their answer options, creation of corresponding bubble sheets, optical analysis and recognition of completed bubble sheets, calculation of final grades based on students' combined results from tests and other activities, and related statistical analysis. Having all these features in a single package positions the *Test* system as an appropriate and effective end-to-end solution for pencil-and-paper knowledge assessment. Compared to its previous major version, described in [1], the new system has been improved in terms of performance, feature set, interoperability with other systems, and ease of use [2, 3]. The *Test* system has been fully developed by the faculty members and students of the ETF, yielding a total of 14 final theses, two international conference papers [1, 3], and ten domestic conference papers on lessons learned and solutions to specific problems.

* Accepted 15 March 2015.

Along with the most interesting results of the previous work on the *Test* system development, this paper includes new contributions. These include comparison of the system with the related state-of-the-art, the key parts of the UML model of the system's new version, a presentation of newer features illustrated with relevant GUI screenshots, performance measurements, and discussion of results obtained through production use of the system.

While using the *Test* system for the introductory programming courses at the ETF (requiring a total of 36 tests each year), a substantial workflow speedup was observed. It was also determined that the automatically assembled tests have higher and approximately three times less varying quality than the tests assembled manually; the quality is quantified by the problem selection criteria, whose parameters are defined by the course teachers. The users of the system have reported positive feedback on the system's behavior.

The following section provides the problem statement, followed by a brief overview of related work. Thereafter, details of the proposed solution are presented. The achieved results are discussed, and the paper concludes with a list of possible topics for future work.

2. Problem statement

Nowadays, computer-based testing is the usual form of knowledge assessment. However, when the number of students exceeds the capacity of available computer facilities, there are two typical solutions. The first is to divide the population into smaller groups and test these groups sequentially. The second is to use pencil-and-paper based testing, ensuring that all students are given the same test at the same time.

The ETF workflow is based on the latter approach. With 560 students enrolled, six examination terms, and three midterms per year, 36 tests must be prepared each year, and the resulting thousands of bubble sheets must be processed. Manual processing of this workload is overwhelmingly tedious and error prone. This was the initial motivation to develop the *Test* system. Other motives include the need for improved quality of individual tests and improved stability of test quality across multiple tests, as well as intention to involve students in development of actual software.

Because learning of programming skills needs both practical and theoretical approach, the assessment process for the introductory programming courses at the ETF uses a kind of mixed assessment methodology described in [4]. Practical programming skills are checked by hands-on programming assignments, coupled with practical examination in the computer lab; mid-term and final exams are administered through knowledge tests. Because actual programming work requires both creation of new code and analysis of existing code, all these knowledge tests typically contain two types of problems. These are problems with coding assignments, where students create their own solutions, and multiple choice questions (MCQs), where students typically need to analyze code. Although MCQs are commonly used to assess the analytic component of knowledge through relatively simple questions, the MCQs used in the tests of the two introductory programming courses are always elaborate: a student has to fully analyze a code snippet in the problem statement, analyze all MCQ options (which typically contain a code snippet), or both. Sometimes, the student must fully synthesize the solution to recognize the correct answer. Because such elaborate MCQs require more time to solve than what usual, simple MCQs would, these tests usually have a relatively small number of MCQs. A typical exam test has two coding problems and six MCQs. To discourage guessing, wrong answers are assigned a negative point value.

The goal of the *Test* system is to support a complete knowledge assessment workflow for large student groups at minimal operational and development cost. The need to keep the operational cost low resulted in a decision to avoid expensive equipment and supplies (e.g., paid pre-printed bubble sheets). An important limiting condition was negligible development funds since the system's inception. At the other end, since the system is to be used in-house, and only by a relatively small user base (i.e., course teachers); there was no need to implement features other than those specifically needed to support the workflow.

Given the above considerations, the features of the system considered important and necessary were the following: manual and automated test assembly using a pool of existing problems and selection criteria; exporting of created tests into printable and on-line formats; creation of multiple versions of the same test by permutation of MCQs and their options; scanning and scoring of bubble sheets; calculation of final grades; and collecting statistics on problem difficulty. Finally, all workflow phases needed to be integrated, so that the output of each phase becomes the input for the subsequent phase.

3. Related work

Related existing solutions that primarily support online assessment can be divided into two major groups: integrated LMSs (Learning Management System) and specialized knowledge assessment soft-

ware. Integrated LMSs, such as *Moodle* [5] or *Blackboard Learn 9.1* [6], support knowledge assessment workflows as a part of a wider learning management workflow. Specialized software, such as *FastTEST* [7], *Respondus 4.0* [8], *QuestionMark Perception* [9], or the web-based application presented in [10] are focused only on assessment, although most specialized solutions provide a feature to export created tests into an LMS. From our perspective, the problem with most existing solutions is that they are primarily intended for online use. In environments where an online use case is not an option, such software can be used only partially: it can create and print out tests and, after the testing and scoring are done, store manually entered results. An exception is *QuestionMark Perception*, which can be coupled with OMR scanners for entry of test results.

Solutions that primarily support pencil-and-paper testing are either rounded ecosystems with custom software (for test creation and result processing) and OMR devices accompanied with paid, pre-printed forms, such as the *Scantron* suite of products [11], or solutions that allow the use of any regular scanner and paper but cover only a part of the assessment workflow, such as *Remark Office OMR 2014* [12]. Solutions in the former group introduce added cost for every testing and are not readily available in all markets. Moreover, when filling out a pre-printed form, each student must mark the test version, which is error prone. Solutions in the latter group typically lack sufficient integration with the test creation phase. Consequently, neither solution class is optimal for the ETF workflow.

Since the test creation phase is very important to us, the work related to the automation of test creation will be covered in more detail. The goal of this phase is to create tests that have similar quality characteristics but different problems. Differences can be introduced by problem selection or by problem generation.

Most solutions described above support test assembly using simple searches, most often based solely on the problem category. To increase the variability of the assembled tests, these solutions can randomize problems being selected, their order, and, for MCQs, the number of options. There are several solutions that offer more than simple random selection based on the problem category. The in-house system *CADAL* [13] controls selection randomness by a precise specification of the selection range (e.g., ''select 19 problems from problems 2 to 77'' or ''include problem 16''). The in-house system eWorkbook [14] organizes problems in a problem repository tree, similar to a file-system structure (allowing nested categories), and allows users to associate metadata with problems; all of this is taken into account during simple random problem selection. The in-house system *Autotest* [15] allows precise specification of test templates using Prolog, where each problem in the template can be specified either as an exact problem or as a problem from a certain category. More sophisticated examples of selection approaches include the use of genetic algorithm with an originally proposed mutation operator [16], as well as genetic algorithms using mate selection proportional to the fitness of the solutions, uniform crossover, survival of the fittest solutions, and a dynamic penalty scheme [17].

Furthermore, psychometric literature [18, 19] presents several other advanced techniques for problem selection, based on mixed integer programming [20], network flow programming [21], simulated annealing [22] or constructive heuristics [23, 24]. *ASC* offers several products (*FastTEST* family, *TestAssembler*) [7], which support test assembly based on the problem difficulty and statistical properties (based on classical test theory or item response theory). *FastTEST* products calculate a test's psychometric characteristics on every test change (problem addition or removal). Although advanced test assembly algorithms based on psychometric theories are undoubtedly valuable for standardized ''high stakes'' and certification testing of large populations, they need problem property metadata to be determined through a trial testing of new problems. These trials are generally very complicated and costly to implement for most of the university courses around the world, and are therefore unfeasible for the ETF.

A well-known method to generate new problems is to use a parameterized template for generation of multiple instances of what is essentially the same problem. Example applications of this approach are QuizPACK [25] (C coding assignments) and LON-CAPA LMS [26] (physics problems). Another method is to process existing materials, and extract problems using natural language processing (NLP) techniques. Regarding the choice of input data, NLP-based solutions rely on manually gathered data [27] or the use of a web-crawler to gather the input data from a well-formatted source such as Wikipedia [28]. Regarding the review of generated problems, one approach is to have the teacher discard inadequate problems after automatic generation has been completed [29] (43% of the generated problems were rejected in that experiment). Another approach is to make the process semi-automatic by having the teacher select the most appropriate distractors (incorrect MCQ options) from the ones suggested by the system [30]. Either

way, a certain level of human interaction is still required at some point of the workflow. Numerous interesting approaches to automatic problem generation are presented in [31].

All presented solutions meet the baseline functionality requirements for their intended purpose. However, several specific functionalities that are needed to optimally support the ETF workflow are not available or not applicable in any of these solutions. For the test assembly phase, we are unaware of a tool with elaborate, yet easily understandable, criteria that would allow teachers to specify certain test properties like those needed for the ETF courses' exams. Most solutions that support automated test assembly either use simple random selection based on a few problem properties or base the problem selection algorithm on tests' psychometric characteristics. All solutions presented so far have their own text editors; using a standalone editor with such a solution requires manual copy-and-paste or a conversion tool such as *WimbaCreate* [32]. In the test grading phase, existing solutions are not applicable in environments similar to the ETF's due to operational costs, local unavailability of supplies (e.g. *Scantron*) or lack of sufficient integration with the test creation phase (e.g. *Remark Office OMR*). A brief overview of knowledge test automatization related regulations and software tools is given in [33].

To conclude, we are unaware of a solution that provides all of the following features: test quality criteria and related user interface that an average teacher can easily understand and manage; automated test assembly using a sophisticated problem selection algorithm; and sufficient integration between test creation and test scoring when multiple test versions, general-purpose text editors, plain paper, and commodity scanners are used.

4. Proposed solution

This section provides an overview of the system structure and description of typical interactions, followed by a brief description of the workflow automation.

4.1 System structure and typical interactions

A typical interaction with the system begins with the *TestBase* component. To create a test, the user (teacher) can filter and select existing problems from a database, or create new ones and add them to the database. At any point, the user can complete the test using the automatic assembly feature. Once the test is prepared, the user can export it and use the *TestMix* component to generate multiple versions of that test, which will then be combined with corresponding bubble sheets into a single, ready to print, *MS Word* document. After students complete the test, completed bubble sheets are scanned and scored by the *TestARS* component. Then, the *TestScore* component can be used to generate problem statistics, as well as student grades in a custom format accepted by the ETF's information system. Other use cases include exporting tests and problems from the *TestBase* database into a SCORM/IMS CP compliant archive or a custom format for later import into a Moodle installation.

Figure 1 shows the four components of the *Test* system directly visible to users (*TestBase*, *TestMix*, *TestARS*, and *TestScore*), one component of the system kernel (*TestCore*), the artifacts generated in

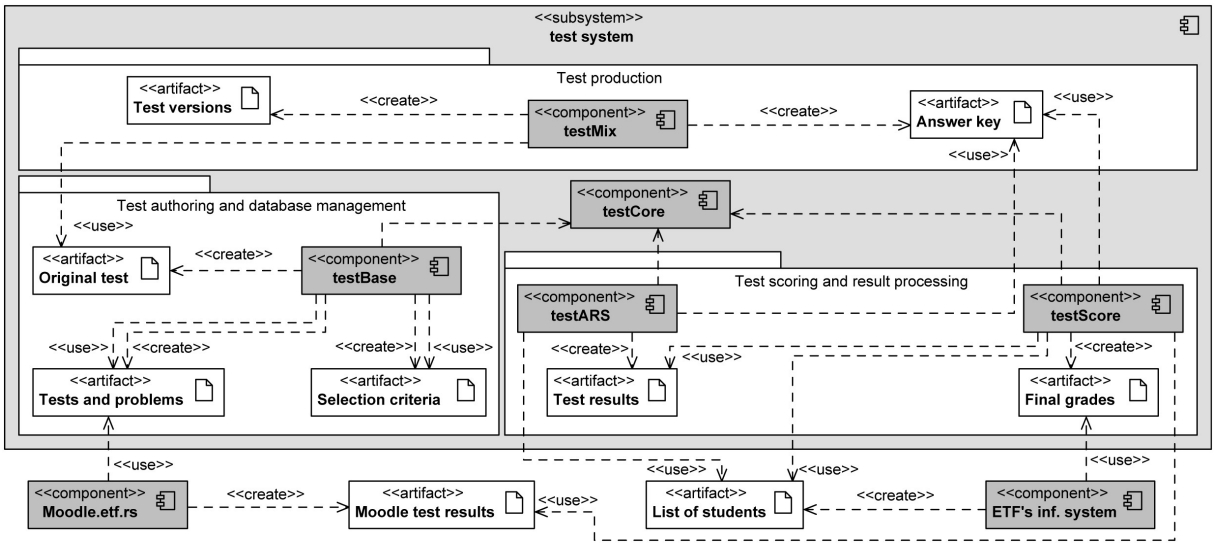

Fig. 1. UML component diagram showing the organization of the *Test* software system and integration with other software systems.

the process, and other related systems (*Moodle* and the ETF's information system). Each of the key functionalities has a corresponding package in Fig. 1. Components are organized in a way such that output of one component is the input of another. Decoupling of the system's components by using artifacts makes them mutually independent, but still well coupled. This allows users to use a single component and to prepare input artifacts outside the system. Moreover, this decoupling makes the system easier to develop and maintain. The central component of the system, called *TestCore*, implements the most important domain abstractions and is used by the other components.

4.2 Workflow automation

The look and feel of *TestBase* resembles contemporary programming environments (e.g., *MS Visual Studio*, *Eclipse*). Figure 2 shows an example use case: the text of the current problem, opened in *MS Word*, is in the upper right part of the screen, while relevant metadata is provided in the surrounding docking windows (panels) that occupy the rest of the *TestBase* window. These panels show available problems, available tests, and the properties of the selected GUI object (course, problem, test, or category). Problems can be filtered by category, difficulty range, creation date range, authors, etc. Tests can be filtered by course only. For the currently selected test, a user can easily manage its contents or automatically complete the test. If any of the problems suggested by *TestBase* is considered inappropriate by the user, it can be marked as rejected; *TestBase* will no longer suggest it during further refinement of the same test. Icons of problems inside a test are colored based on acceptance decisions (green—accepted, red—rejected, yellow—decision pending). Similarly, color coding is also applied to test icons according to their conformance to the associated problem selection criterion (green—conforming and complete, yellow—conforming but needs more problems to be complete, red—not conforming, dark green—test without criterion, white—empty test). Moreover, a test's property page will show that test's compliance to all associated selection criteria and show the non-compliant properties in red.

Figure 3 details how *TestBase* internally represents test states and how user actions trigger state transitions. For example, adding a problem to an empty test (white icon) can have multiple outcomes. If the test has a criterion assigned, the quality function will be calculated after this addition. In the case of zero quality, the test will go into the "Unacceptable" state (red icon). Otherwise, the test will go into the "In progress" state (yellow icon) or, in the special case that the criterion requires only one problem, into the "Acceptable" state (green icon). If the test has no criterion assigned, the test will go into the "Acceptable" state (dark green icon).

Figure 4 shows relations between the key entities of *TestBase* logic. Decoupling problems from spe-

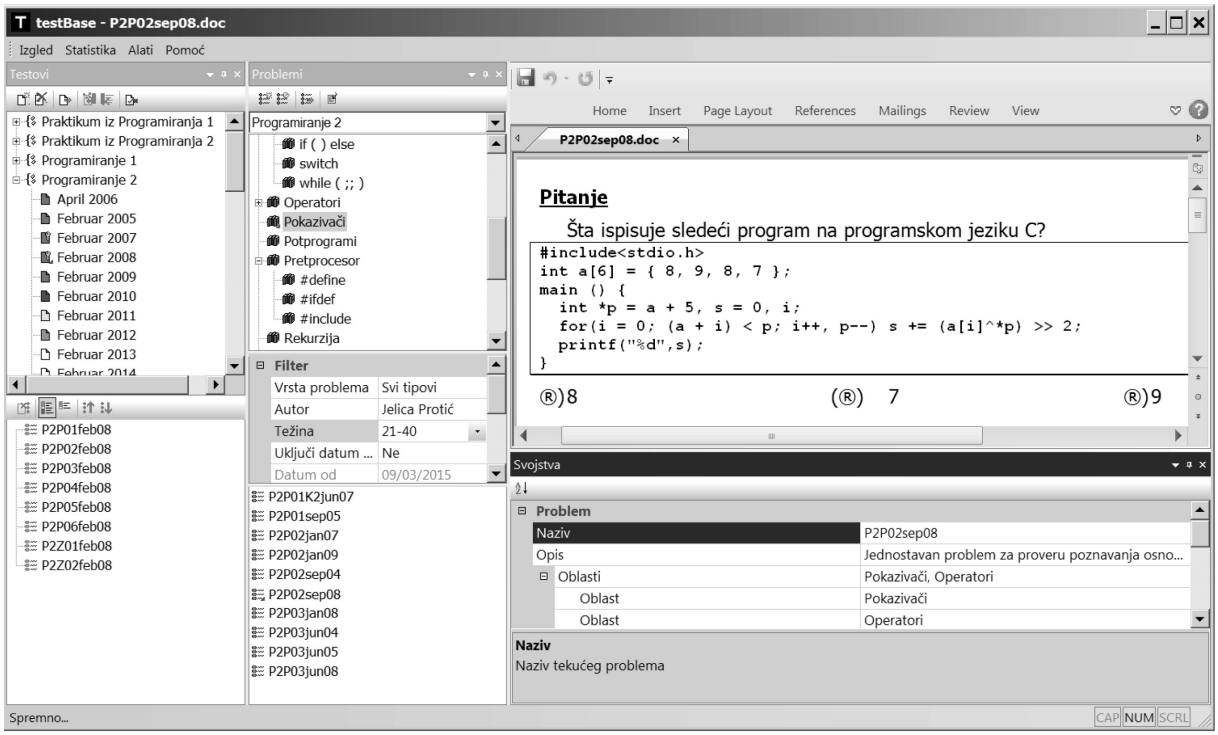

Fig. 2. Screenshot of *TestBase* with all panels and problem text opened in an embedded *MS Word* window.

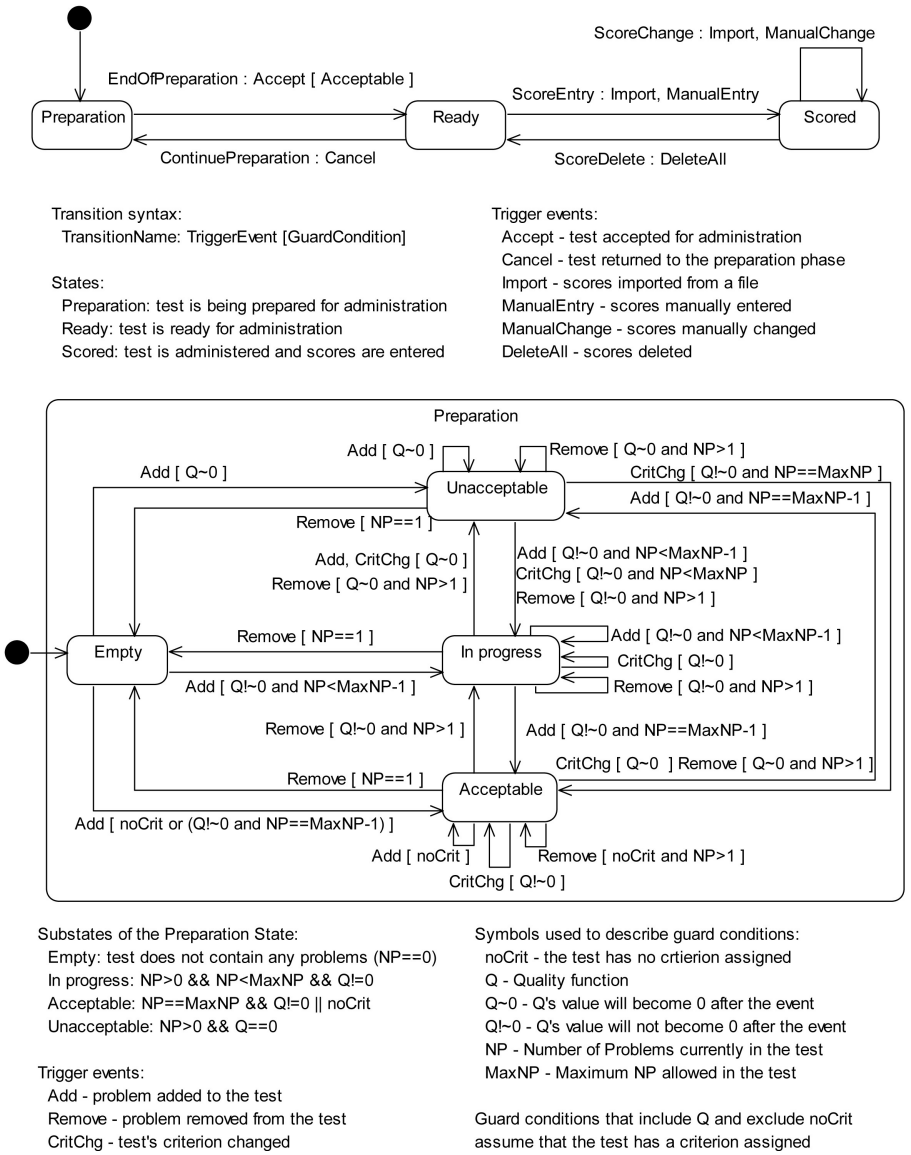

Fig. 3. State diagrams showing possible states of a test and relevant transitions based on user's actions. (Names of transitions between sub-states of the Preparation state omitted for brevity.)

cific courses allows for any problem to be used in more than one course. An example is a main course and its corresponding practicum. Another example are courses with overlapping contents, typically seen during educational reforms (when the new curriculum partially overlaps the old one) or when the same material is taught at different levels across study programs. To ensure that problems within a test are appropriate for that test's course, *TestBase* enforces three constraints. First, during editing of a selection criterion for a given course, *TestBase* will only show categories associated with that course. Second, when the user is associating a test with a criterion, *TestBase* will offer only criteria associated with that test's course. Third, during automatic assembly of a test for a given course, *TestBase* will consider only those problems that have all their categories associated with that course. Categories are hierarchically organized through parent-child composition.

An important differentiating feature of *TestBase* is its approach to automated test assembly. The problem selection algorithm used in *TestBase* is based on a rich set of formalized problem selection criteria. These criteria are designed to resemble those that we intuitively use. They can relate to an individual problem in a test, to a group of problems, or to the test as a whole. The criteria consider acceptable problem categories, problem difficulty, and problem usage history. Figure 5 shows the dialog used to manage these criteria. The course selection combo-box and named criteria associated

to the current course are shown on the left section of the dialog. Parameters of the current test criterion are shown in the middle section, including the list of criteria specific to a particular problem type (KVP). The right section shows parameters of the current KVP in a compact property grid. Default values of all parameters are set so that they do not eliminate any problems during the search. Detailed description of criteria is available in [1, 2].

The problem selection algorithm is based on a modified hill-climbing search algorithm: on selection of the next search state (a search state is defined as a group of problems selected to be included in a test; a transition to the next state is defined by adding the next problem to the group), the algorithm randomly selects one of the N best candidate states instead of the best one. This approach allows for different results in subsequent runs.

Some eliminatory criteria are applied before the search is started to prevent selection of unacceptable problems in the search process, hence reducing the search space. States that contain k problems from n acceptable problems are generated as k-combinations without repetition. A problem can be added to a state only if that problem's unique identifier is greater than identifiers of all the problems already present in the state. Each selection criterion used during the search has a heuristic function and a weight factor associated with it. To evaluate a search state, a quality function is applied to its group of problems. The algorithm will remove any zero quality state from the search space immediately after state's creation. An early version of the problem selection criteria and their classification were originally presented in [1], while the elaborated version is presented in [2]. Details of the current implementation of the automated test assembly functionality are given in [2], as well.

Figure 6 shows an example run of the algorithm for the case when it needs to assemble a three problem test from nine eligible problems. In step 1, it will create groups (states) with one acceptable problem per group and calculate each group's quality. Given the above mentioned constraint to generation of combinations of problems, one-problem states {8} and {9} are not generated because they cannot be used to complete assembly of a three-problem test. In step 2, the algorithm will sort newly created states by quality. Next, it will determine the number of acceptable search states, AS. An acceptable search state is one with quality within the allowed quality margin, QM, when compared to the state with the best quality. Another user-defined search parameter is search range, SR, defined as the maximal number of states to be considered when selecting the next search state. If the number of candidate states with maximal quality is larger than SR, then SR will be appropriately enlarged to cover all equal states for that particular step. The next search state gets selected from S candidate states, where S is defined as minimum of AS and SR. Setting $QM = 0\%$ or $SR = 1$ makes the algorithm behave like the standard hill-climbing algorithm; it will pick the best possible candidate state on each state transition. In Fig. 6, there are five acceptable states within the quality margin $QM = 20\%$, but only the first four are considered in step 2 due to the value of SR = 4.

In step 3, one of those four states is randomly selected and transitioned into all possible next states. For the same reason that states {8} and {9} were not generated in step 1, the state {3, 9} is not generated in step 3. In step 4, one state is randomly selected from the three acceptable states (the fourth state is outside of the quality margin) and transitioned further. In step 5, since the states are completed with the required number of problems (three), only the best is saved as the test proposal, while others are discarded. In step 6, the search continues backward to remaining states with two problems; one of the two remaining acceptable states is randomly selected. In step 7, a group of three problems better than the current best is found; hence the test proposal is updated with it. The process continues until the entire search space has been traversed, the duration of the search reaches the user-defined timeout, or the user stops the search.

Another interesting characteristic of *TestBase* is that it uses external software for problem editing. In order to enable users to edit their problems in a

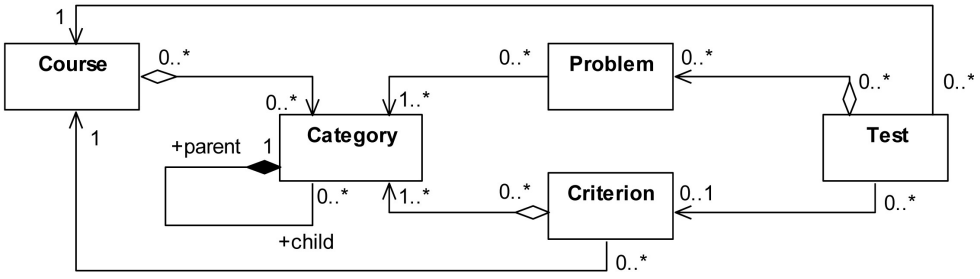

Fig. 4. UML class diagram showing relations between most important concepts in *TestBase*.

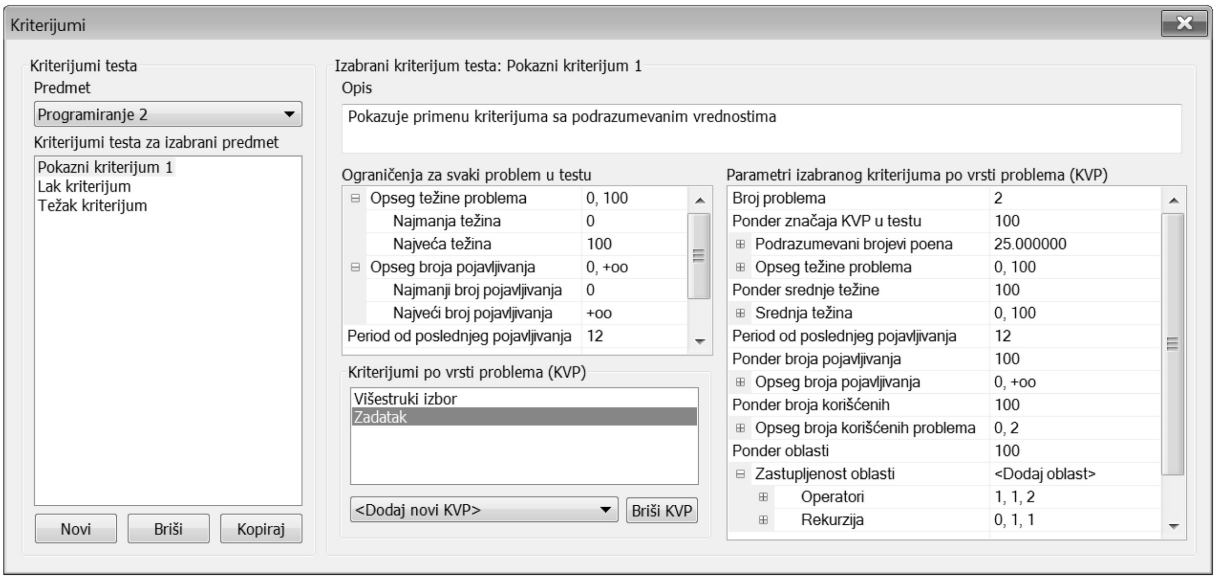

Fig. 5. Screenshot of the dialog for criteria management in *TestBase*.

familiar environment, *TestBase* uses *MS Word* for editing and printing. Modular design of *TestBase* allows for relatively easy use of other file formats and editing software.

The capability to export to *Moodle* comprises two distinct software components. The first is the *TestBase*'s functionality that exports the test and problem data in a simple, interchange, XML-based format. The second is a custom *Moodle* module that enables importing the data exported by *TestBase* into a given *Moodle* installation database. After the import is completed, there is no significant difference between data created in *TestBase* or in *Moodle*. This enables the user to fully utilize *Moodle*'s assessment functionality on *TestBase* data. The custom module maintains a mapping between *TestBase* and *Moodle* IDs for all exported problems and tests. This enables easy updating (within the *Moodle* installation database) of already exported problems from *TestBase*.

To make cheating on a pencil-and-paper test harder, the *TestMix* component enables preparation of different versions of the same test. It takes the test that needs to be "shuffled" and generates a document with multiple versions of that test. Moreover, for each version of a test, *TestMix* can prepare

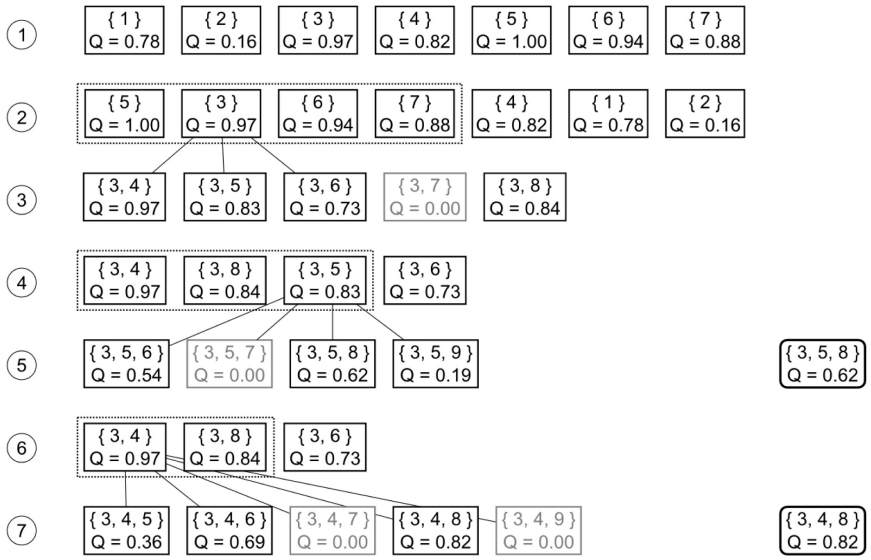

Fig. 6. Example run of the algorithm when it needs to select three out of nine problems, with quality margin $QM = 20\%$ and state range $SR = 4$; zero quality states are colored light gray.

a corresponding bubble sheet with appropriate number of rows for answers and a visually encoded version identifier. Everything that needs to be printed (test, bubble sheet, instructions, answer keys) is arranged in a single ready-to-print document.

After the pencil-and-paper test administration phase is done, the *TestARS* component can be used for optical recognition and scoring of the answers to the MCQ problems. Input to *TestARS* can come from a TWAIN-compatible scanner or from a file (currently supported formats are BMP and TIFF). Bubble sheet scans are grouped together using a project file, which typically aggregates all scans from one testing. The output of *TestARS* contains student ID, test version, student answers, and corresponding scores of each student.

Figure 7 shows the main screen of *TestARS*. The left part shows collection of sheets belonging to the current project, represented in a tree structure. Each sheet is represented as a leaf; non-leaf nodes are used for grouping of logically-related sheets (for example, grouping of sheets from one classroom when testing is done in several classrooms). The middle part shows the scanned sheet; the bordered area, which is analyzed, contains the student's ID at the top, the answer area in the middle, and the binary representation of a test version code at the bottom. The user can turn on visual indicators showing detected answers in an appropriate color and the detected test version bits. The right part shows the results of analysis: student's ID number, name, test version, and acquired scores (total and particular for each MCQ).

The analysis algorithm is resilient to planar distortions of the sheet scan, typical for handheld scanners. If the user observes an analysis error (which is always possible but very infrequent in practice), he/she can manually correct it or repeat the analysis with adjusted settings.

The settings of sheet dimensions, bubble coordinates, and all other optical mark recognition parameters are done in a separate window. The current sheet is used as a visual reference, with all relevant analysis settings shown in a user-controlled overlay; while changing the settings using this overlay, the user has immediate visual feedback on whether the settings being updated are appropriate for the current sheet. Correct answers can be loaded from files produced by *TestMix* or entered manually.

The *TestScore* component calculates individual student results, and based on aggregate results it also calculates parameters of the problems included in a given test. Using this component, a student's test score can be combined with the student's results on other activities that contribute to the final grade, as defined by the course's grading rules, defined in a plain text file. Once calculations are done, *TestScore* can generate a report of the students' results and final grades, formatted as per ETF's information system's requirements. Because the component is primarily intended to test the system's kernel *TestCore*, it is still at a proof-of-concept level.

5. Discussion of results

All results in this section are from the system's use at the ETF, where it supports the knowledge assessment workflow for two Programming courses and, to a certain extent, their corresponding practicum courses. System performance was measured on a configuration with an Intel Core i5 at 3.20 GHz and 4GB of DDR3-10700 (667MHz) memory.

Table 1 illustrates reliability of the answer recognition algorithm used in *TestARS*, aggregated from the empirical data during the last ten years of production use. Each *TestARS* user scanned thousands of bubble sheets and reported similar findings on the observed error rates. Error rates for the handheld scanner use case would be significantly higher if *TestARS* did not have an algorithm to correct distortion inherent to handheld scanners. When a scanner equipped with an automatic document feeder (ADF) is used, there are no complaints from students about the accuracy of recognition results. Experience shows that 150 dpi is a sufficient resolution for the recognition needs and that error rates stay the same at higher resolutions.

Table 2 shows the processing speed of the recognition algorithm itself. With an example office scanner that has throughput of 30 pages per minute, the total theoretical throughput with 150dpi resolution would be around 1700 bubble sheets per hour. This calculated speed is similar to the speed of low-end commercial OMR readers, such as *Scantron iNSIGHT 20*.

Table 3 shows the results of a worst case analysis of the test assembly tool's performance, when whole search space is traversed. The table shows elapsed time for different values of target number of problems in the test (*NP*). Searches were performed on a series of 36 candidate problems. In order to ensure that the entire search space gets traversed, these problems had their properties artificially set to satisfy all assembly criteria. We measured the elapsed search time and the number of acceptable proposals assembled during the search (*NAP*). On the other end, the number of possible proposals (*NPP*) was calculated as the value of the binomial coefficient of the number of candidate problems (NP_{cand}) and *NP*. The fact that the *NAP* and the *NPP* in this experiment were always the same means that the entire search state space was traversed in every run. For most real-life examples, where many

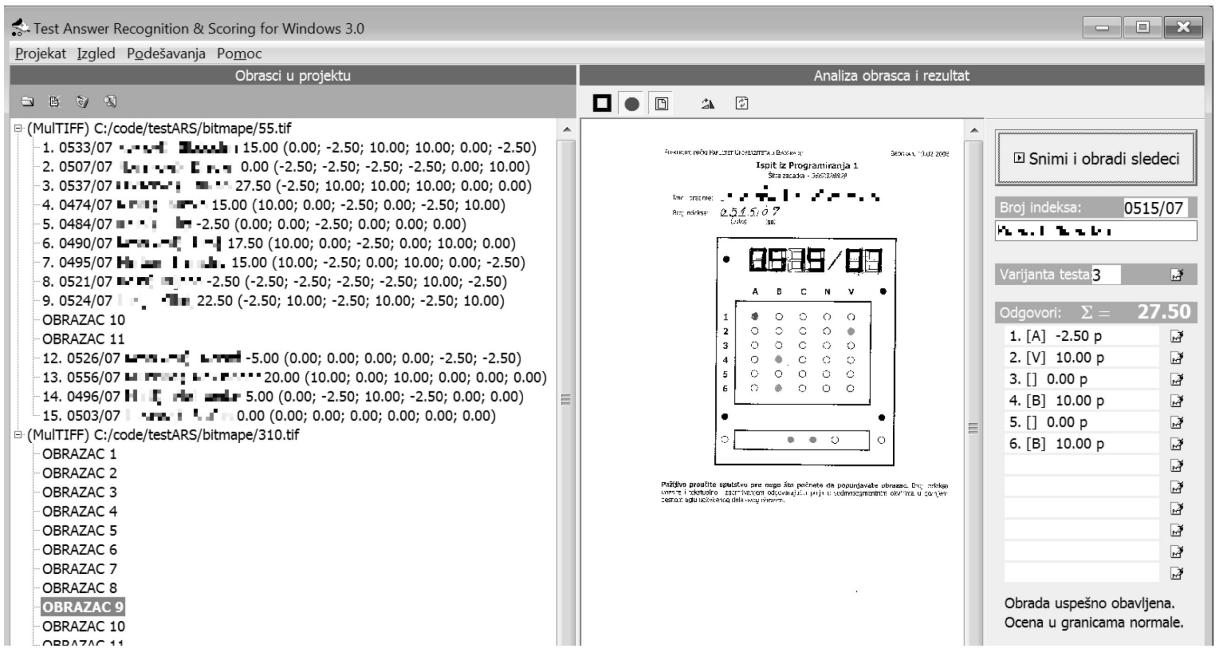

Fig. 7. Screenshot showing *TestARS* after completed bubble sheet analysis. (Lower part cropped. Identities obscured for privacy.)

search states are eliminated early in the search and tests have up to 20 problems, a test is assembled in a matter of seconds. An assembly of a typical test (two problems, six MCQs) on the average takes one second. This is negligible when compared to time needed to perform the ''mix-up'' and print preparation phases, which take around 10 seconds because they are implemented as *MS Word* macro procedures.

In order to quantify the difference between manually and automatically assembled tests, a comparative experimental analysis of test quality was conducted. First, the criteria that resembled those used by the course's teachers were entered into the content database. Next, those criteria were associated with a set of manually prepared final tests of one of the Programming courses; this set included all final exam tests over a period of four school years. Finally, all the tests from the set were temporarily removed from the content database, and automatic assembly was performed using the same problems. The difficulty levels of problems in the database were determined based on a statistical analysis of successful and unsuccessful student responses. Table 4 summarizes the results of this experiment, which showed that use of the automated assembly feature is feasible, and that automatically assembled tests have slightly higher quality and approximately three times lower standard deviation of quality than those assembled manually.

For the measurement presented in Table 4, we used the tests assembled manually by the courses' staff. Test quality is calculated by a quality function, as described in [1]. The most important parameters that affect a test's quality are individual and aggregate problem difficulty, number of previous uses of a particular problem, and intended number of problems for each category.

The Export-to-*Moodle* feature was tested for proof of concept on elective practicum courses tied to the respective Programming courses. The practicum courses require students to solve programming assignments in a computer lab. In each school year, there are 120–150 students enrolled in these courses; this number is roughly a quarter of students enrolled in Programming 1 and Programming 2 courses. In each lab exercise, students were asked to go over a test composed of ten relatively simple problems of several types (MCQ, short-

Table 1. Answer recognition error rates gathered through production use of *TestARS*

Report method	Handheld scanner	ADF scanner
Users (before results are published)	<1%	<0.3%
Students (based on published results)	<0.1%	0%

Table 2. Average *TestARS* processing times dependent on the scan resolution

Scan resolution [DPI]	Average processing time per sheet [s]	Standard deviation [s]
150	0.11	0.02
200	0.19	0.04
300	0.37	0.06

answer, coding assignment). Problems were selected randomly by *Moodle's* own testing functionality from a larger pool of problems, all of which had been imported from *TestBase*. Grading was done manually for coding assignments, and automatically for other problem types. Students were unanimously positive about this addition to practica's curricula. They reported that frequent formative assessments help them better understand where their programming knowledge can be improved. On the other end, the teaching staff reported that analysis of MCQ statistics and the students' answers to open-ended problems helped them to understand the most common topics where students needed more explanations or practical work.

Although the primary goals of the *Test* system's development were to make exams more objective and to facilitate the teacher's job by saving time and reducing error rates, while working towards these goals another important result was achieved: multiple final year students got exposed to the development of real production software for their final thesis. Almost every student had to master at least one new technology before starting with his or her actual thesis work; also, a typical student would become very productive approximately at the same time when his or her thesis work was completed. We believe that educational benefits greatly compensate for some of the disadvantages, as all those students have reported that the experience gained greatly helped them in their future projects. Also, several of those students have participated in writing conference papers related to their thesis work, together with authors of this paper who were their advisors. On the other hand, the problem that greatly delayed the system's development was temporal discontinuity between parts of the job done by students from different generations. Those delays introduced changes to the requirements, development platform, and subsequent integration of the system's parts. To mitigate risks caused by temporal discontinuity and frequent developer changes, the usual good software engineering practices were employed: all issues were tracked using an issue tracker system, coupled with a revision control system; functional specifications of system components were always made integral parts of relevant final theses; and ideas for future work were prepared as propositions for future theses and offered to final year students.

Finally, it should be noted that in the current development phase, the system is intended to be an in-house prototype for only a small and limited user base, rather than a polished product. Therefore, some of the functionalities needed for wider usage are either only partially implemented (additional problem types, statistical analysis of test results, installation package, integration with contemporary distance learning tools, etc.) or not implemented at all (globalization, concurrent user access, deployment strategy, advanced information security, etc.). The developed question selection algorithm is currently parallelized only per item type, so the implementation doesn't optimally exploit available parallelism of modern CPUs and GPUs. Although comparable to dedicated OMR readers, the OMR algorithm has room for improvement in terms of processing time and error rate.

Table 3. The dependence of elapsed search time on NP, measured for $NP_{cand} = 36$.

NP	NAP	NPP	Time (s)
7	8,347,680	8,347,680	12
8	30,260,340	30,260,340	47
9	94,143,280	94,143,280	157
10	254,186,856	254,186,856	461
11	600,805,296	600,805,296	1,142
12	1,251,677,700	1,251,677,700	2,527
13	2,310,789,600	2,310,789,600	4,946
14	3,796,297,200	3,796,297,200	8,704

Table 4. Durations of preparation phases for a typical Programming exam

	Manually assembled tests	Automatically assembled tests
Mean test quality	0.934	0.952
Standard deviation	0.069	0.023

6. Conclusions

The *Test* system is essentially a research and educational prototype of an assessment system. The system is primarily developed to be used at the ETF as a tool that increases speed and reliability of knowledge test preparation, scoring, and result processing workflow. Additionally, it presents a training ground for undergraduate and graduate students' final theses, as well as for research in the area of authoring tools. Due to its well-rounded feature set, built upon and proven by long-term experience in an environment with computer resources insufficient for large groups of candidates, it can be considered a feasible solution for similar environments.

This paper encompasses results produced during the long period of the *Test* system's development and experience of its in-house production usage. The paper's focus is at previously unpublished contributions, like the key parts of the system's design model, the GUI for formulating of the test assembly criteria, and the functioning of the modified hill-climbing algorithm.

Possible future work can be roughly divided into two categories: research and implementation.

Research-wise, new problem selection algorithms based on neural networks and genetic algorithms, data mining, or linear programming, could be explored and added to the system. Implementation-wise, the existing feature set can be extended with new problem types, parameterized problems, multiple user and globalization support, more detailed result analysis with problem difficulty feedback, self-assessment features such as adaptive problem selection tailored to each student's exhibited knowledge following the concept of personalized learning [34], and the use of high-performance computing to speed up the problem selection algorithm. Furthermore, the system's development platform could be unified and ported to some of the contemporary Web development platforms.

Acknowledgements—We would like to thank former students Danka Lolić, Miodrag Krunić, Predrag Cerović, Petar Opačić, Darko Popović, Vladan Gunjić, Goran Anucojić, Marko Krstović, Dušan Stojanović, Predrag Jovanović, Dejan Rizvan, Branko Kokanović, Nikola Milošević, and Aleksandar Đurić for their contribution to development of the *Test* system, as well as our colleagues Đorde Đurdević, Miloš Cvetanović, Zaharije Radivojević, Marko Mišić, and Igor Andelković for providing valuable feedback that made the final result considerably better.

We would also like to thank Dr. Santanu Dutta and Matthew A. Kenny for valuable comments and suggestions.

Work of Igor Tartalja and Dragan Bojić was partially supported by the projects TR32039 and TR32047 of the Ministry of Science and Technological Development of the Republic of Serbia.

References

1. J. Ž. Protić, D. B. Bojić and I. I. Tartalja, *test*: Tools for evaluation of students' tests— a development experience, in *Proc. 31st ASEE/IEEE Frontiers in Education Conference*, Reno, NV, USA, 2001, pp. F3A-6–F3A-12.
2. A. M. Bošnjaković, Development of software system for authoring, scoring, and result processing of knowledge tests, Magister thesis (in Serbian), Univ. Belgrade—School of El. Eng., Serbia, 2010.
3. A. M. Bošnjaković, J. Ž. Protić and I. I. Tartalja, Development of a software system for automated test assembly and scoring, *Proc. Int'l Conf. of Education, Research and Innovation (ICERI'10)*, Madrid, Spain, 2010, pp. 6012–6016.
4. D. Diaz, T. J. Leo, E. Mora and J. A. Somolinos, Mixed Assessment Methodology in Engineering Higher Education based on Quality Control Concepts, *International Journal of Engineering Education*, **30**(2), 2014 pp. 424–437.
5. Moodle Community, Moodle, www.moodle.org. Accessed 09.11.2014.
6. Blackboard, Washington, DC, Blackboard Learn 9.1, http://www.blackboard.com/platforms/learn/overview.aspx. Accessed 09.11.2014.
7. ASC, St. Paul, MN, Test Development Software, http://assess.com/xcart/home.php?cat=28. Accessed 09.11.2014.
8. Respondus, Inc., Redmond, WA, Respondus 4.0, http://www.respondus.com/products/respondus/. Accessed 09.11.2014.
9. QuestionMark, Norwalk, CT, QuestionMark Perception, https://www.questionmark.com/us/perception/Pages/default.aspx. Accessed 09.11.2014.
10. G. Martinović and B. Zorić, Web Application for Knowledge Assessment, *International Journal of Engineering Education*, **30**(4), 2014, pp.779–787.
11. Scantron Corporation, Eagan, MN, ParSystem Integrated Testing Suite, http://www.scantron.com/software/classroom-testing/parsystem/overview/. Accessed 09.11.2014.
12. Gravic, Inc., Malvern, PA, Remark Office OMR, http://www.gravic.com/remark/officeomr/. Accessed 09.11.2014.
13. A. Carbone, P. Schendzielorz and J. D. Zakis, Electronic assessment and self-paced learning on the Web using a multiple choice quiz generator *International Journal of Electrical Engineering Education*, **37**(2), 2000, pp. 119–125.
14. G. Costagliola, F. Ferrucci, V. Fuccella and R. Oliveto, eWorkbook: a Computer Aided Assessment System, *International Journal of Distance Education Technologies*, **5**(3), 2007, pp. 24–41.
15. F. Hernández-Del-Olmo and E. Gaudioso, Autotest: An educational software application to support teachers in creating tests, *Computer Applications in Engineering Education*, **21**(4), 2013, pp. 636–640.
16. M. Yildirim, A genetic algorithm for generating test from a question bank, *Computer Applications in Engineering Education*, **18**(2), 2010, pp. 298–305.
17. A. J. Verschoor, Genetic Algorithms for Automated Test Assembly, Ph.D. dissertation, Dept. Research Methodology, Measurement and Data Analysis, Univ. Twente, Netherlands, 2007.
18. W. J. van der Linden, *Linear models for optimal test design*, ch. 5–9, 2005, Springer, New York, NY.
19. H. A. Huitzing, B. P. Veldkamp and A. J. Verschoor, Infeasibility in Automated Test Assembly Models: A Comparison Study of Different Methods, *Journal of Educational Measurement*, **42**(3), 2005, pp. 223–243.
20. W. J. van der Linden, B. P. Veldkamp and L. M. Reese, An integer programming approach to item pool design, *Applied Psychological Measurement*, **24**(2), 2000, pp. 139–150.
21. R. D. Armstrong, D. H. Jones and Z. Wang, Network optimization in constrained standardized test construction, in K. D. Lawrence and G. R. Reeves (ed.), *Applications of Management Science: Network Optimization Applications*, **8**, pp. 189–212. Greenwich, CT: JAI Press, 1995.
22. B. P. Veldkamp, Constrained multidimensional test assembly, *Applied Psychological Measurement*, **26**(2), 2002, pp. 133–146.
23. R. M. Luecht, Computer-assisted test assembly using optimization heuristics, *Applied Psychological Measurement*, **22**(3), 1990, pp. 224–236.
24. L. Swanson and M. L. Stocking, A model and heuristic for solving very large item selection problems, *Applied Psychological Measurement*, **17**(2), 1993, pp. 151–166.
25. P. Brusilovsky and S. Sosnovsky, Individualized exercises for self-assessment of programming knowledge: An evaluation of QuizPACK, *J. Educational Resources in Computing*, **5**(2), 2005, pp. 1–22.
26. D. A. Kashy, G. Albertelli, G. Ashkenazi, E. Kashy, H.-K. Ng, and M. Thoennessen, Individualized Interactive Exercises: A Promising Role for Network Technology, in *Proc. 31st ASEE/IEEE Frontiers in Education Conference*, Reno, NV, USA, 2001, pp. F1C-8–F1C-13.
27. N. Afzal, Unsupervised Relation Extraction for E-Learning Applications, Ph.D. dissertation, Research Inst. for Inf. and Language Processing, Univ. Wolverhampton, UK, 2012.
28. S. Agrawal, Automatic Quiz Generator, M.Sc. thesis, Dept. Comp. Science, Univ. Sheffield, UK, 2011.
29. R. Mitkov and L. A. Ha, Computer-aided generation of multiple-choice tests, in *Proc. HLT-NAACL 03 workshop on Building educational applications using natural language processing*, Morristown, NJ, USA, 2003, pp. 17–22.
30. A. Hoshino and H. Nakagawa, A Cloze Test Authoring System and its Automation, in *Advances in web based learning—ICWL 2007 (Proc. 6th Int'l Conf. Edinburgh, UK, August 15–17, 2007, Revised Papers)*, H. Leung, F. W. B. Li, R. W. H. Lau, and Q. Li, Ed., New York: Springer, 2008, pp. 252–263.
31. M. J. Gierl and T. M. Haladyna, Ed., *Automatic Item Generation: Theory and Practice*, Routledge, New York, NY, 2013.
32. Wimba, Inc., New York, NY, WimbaCreate, http://www.wimba.com/products/wimba_create. Accessed 09.11.2014.

33. A. Bošnjaković, I. Tartalja and J. Protić, Support for Knowledge Tests: Brief Summary of Regulations and Software, *The IPSI BgD Transactions on Internet Research*, **3**(1), 2007, pp. 25–29.

34. A. Staikopoulos, I. O'Keeffe, R. Rafter, E. Walsh, B. Yousuf, O. Conlan and V. Wade, AMASE: A framework for supporting personalised activity-based learning on the web, *Computer Science and Information Systems*, **11**(1), 2014, pp. 343–367.

Andrija M. Bošnjaković received the B.Sc. and M.Sc. degrees in electrical engineering and computer science from the ETF. He is currently a software consultant. His affiliation at the time when the work described in this paper was performed was the same as of the other three authors. His current professional interests include object-oriented and parallel programming, as well as cloud computing technologies and stereoscopic technologies.

Jelica Ž. Protić received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from the ETF. She is currently an Associate Professor of computer engineering and informatics with the ETF. Her research interests include engineering education and educational tools, distributed systems and all aspects of computer-based quantitative performance analysis and modeling. With Milo Tomašević and Veljko Milutinović, she coauthored Distributed Shared Memory: Concepts and Systems (IEEE CS Press, 1997). She was Vice Dean of Education from 2004 until 2009, and she currently serves as a Dean's Counselor for quality assurance and accreditation.

Dragan M. Bojić received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering and computer science from the ETF. He is currently an Assistant Professor with the ETF. His research interests include software testing, effort estimation, and e-learning technologies.

Igor I. Tartalja received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from the ETF. From 1984 to 1989 he was with the Laboratory for Computer Engineering, Institute for Nuclear Sciences, Vinča. In 1989 he joined the ETF. He is coauthor of one IEEE CS book and over 60 scientific papers. He contributed to more than 20 research and development projects. He led a dozen of them, the results of which are commercially applied. He received two national awards for contributions in the software engineering area. He is currently an Associate Professor with the ETF, teaching subjects on Object-oriented programming, Software design, and Computer graphics. His current research interests include object-oriented modeling and programming, computer graphics, geoinformatics, and edutainment.