**Enthusiast:** An authoring tool for automatic generation of paper-and-pencil multiple-choice tests

Jan Šnajder, Marko Čupić, Bojana Dalbelo Bašić, Saša Petrović

Faculty of Electrical Engineering and Computing, University of Zagreb

**Key words:** knowledge assessment, multiple-choice questions, authoring tool, examination generation software, large class assessment support

**Abstract:**

In this paper we describe Enthusiast – a flexible tool for the automatic generation of pen-and-pencil multiple-choice test sheets. Enthusiast uses a database of multiple-choice questions and a test specification provided by the user to generate randomized multiple-choice test sheets suitable for machine-scoring. The questions database may be augmented with metadata tags, effectively defining user-specific questions taxonomy upon which detailed test specifications can be based. Enthusiast may be used to generate test sheets of adequate variability, speed up test administration, and ensure objective and fast grading. We have recently used Enthusiast in several courses at our faculty, for both summative and formative knowledge assessment, and received positive feedback from students.

**1 Introduction**

Assessing student knowledge can be a challenging task, especially for courses with large enrolments. Open-ended question tests (e.g., short answer questions and essays) are perhaps the most simple to create, but grading them is very tedious and time consuming. If several hundred students have taken the test, results might be available only several weeks later, leaving students with no immediate feedback. More importantly, consistent grading with this type of test is difficult to achieve. In order to provide objective and consistent assessment of students' knowledge, as well as more time efficient grading, multiple-choice tests are often used. Multiple-choice test are considered a valid alternative to open-ended tests; in [1] it is even argued that “open-ended questions should be used solely to test aspects that cannot be tested with multiple-choice questions.” This is especially true if questions are designed according to established guidelines, such as the ones suggested in [2].

Multiple-choice tests may be used for many forms of knowledge assessment, such as formative ones (e.g., end-of-lecture quizzes) and summative ones (e.g., written mid-term and final exams). If the number of students is relatively small, this kind of test can be held in computer equipped classrooms under teacher supervision. Unfortunately, with a large number of students this kind of knowledge assessment is often not feasible. An alternative to this are the traditional paper-and-pencil multiple-choice tests. Because multiple-choice tests can be machine-scored, significant time savings are gained over free form-tests. However, when administering multiple-choice paper-and-pencil tests in a classroom, special care must be taken to prevent test cheating. This typically means that the test must be prepared in a number of test variants. Ideally, test variants should differ not merely in the presentation order of
questions and answers, but also feature slightly modified questions. Preparing such test variants manually is an extremely error prone task.

To address the above mentioned issues, we devised Enthusiast – an authoring tool for the automatic generation of pen-and-pencil multiple-choice tests. Enthusiast uses a plain-text database of multiple-choice questions and a test specification provided by the user to generate randomized multiple-choice test sheets suitable for machine-scoring. The questions database may be augmented with metadata tags that describe the topic and the type of each question, effectively defining user-specific questions taxonomy upon which detailed test specifications can be based. The variability across test sheets (the extent to which questions and answers differ across test sheets) can also be adjusted. Enthusiast generates test sheets in LaTeX format, a widely used document preparation system of high typographical standard [3]; the format of the sheets can of course be customized to suit specific needs.

The rest of the paper is structured as follows. In next section, we describe Enthusiast in more detail, while in Section 3 we discuss practical experience with Enthusiast. A brief comparison of Enthusiast with other similar systems is given in Section 4. Section 5 concludes the paper and briefly explains future work.

2 Enthusiast

Enthusiast is a stand-alone tool implemented in the functional programming language Haskell [4]. It uses a questions database and a test specification as input, and generates randomized test sheets as output.

2.1 Questions Database

Contrary to most existing examination generation software, Enthusiast uses a questions database encoded in a simple plain-text format rather than a full-blown database or XML files. The questions database may be organized into several files and folders. The main motivation behind this is that question editing should be kept as simple as possible, allowing the user to focus on content instead of the form. In order to improve the variability across test sheets, each question in the database may be complemented with mutually exclusive question variants, as well as a redundant number of correct and incorrect answers. Moreover, each question may be associated with user-specific tags (metadata keywords) that describe its topic and type. This effectively allows for user-specific questions taxonomy. Moreover, tags may be organized hierarchically, allowing for a more fine-grained and more comprehensive taxonomy. Based on the tag metadata, the user can provide Enthusiast with a detailed test specification defining the content and type of the test.

An excerpt from question database is given in Fig. 1. The questions are on Artificial intelligence and the problem of state-space search. In the question database file, user’s comments are prefixed by a percent sign, while each question is labelled with a unique identifier prefixed by the '@' sign. This example features two questions: question @1 (lines 6 through 34) and question @2 (lines 36 through 47). Question @1 comes in two variants (lines 8 through 20 and lines 22 through 34). Each question consist of the question text followed by a number of answer options; the correct options are prefixed by the '+' sign and the incorrect ones by the '–' sign. The required number of options is determined by the test type; e.g., a one-out-of-four test requires a minimum of one correct and three incorrect answer options. If a larger-than-minimum number of correct or incorrect options are provided, Enthusiast will
randomly choose the required number of answer options. Options that for some reason or other are preferred are marked with the ‘!’ sign; these are the options that Enthusiast will consider choosing first. A default option, marked by an ‘*’ sign, is a sort of back-off option and will be presented last.

--

In Fig.1, the user-specific tags are shown in bold. To make tagging less tedious, the tags in the file may be specified at three different levels, each of progressively narrower scope. At the top-most level, tags that are common to all questions in the file are specified (line 4). At the second level, each particular question is tagged (lines 6 and 36). Finally, at the third level, each particular question variant may be tagged (lines 8 and 22). Hierarchical relationship between tags is indicated with a colon. For example, tags search, blind, and depthFirst may be written as search:blind:depthFirst to reflect the fact that blind search is a kind of search procedure and that depth first search is in turn a kind of blind search. To make hierarchical tagging more convenient, we allow hierarchical tags to be broken down into parts and specified incrementally. This is accomplished by propagating tag specifications to lower scopes: a tag ending with a colon will be propagated to the immediate lower scope, while tags

```plaintext
$Course$: Artificial intelligence
$Lecture$: State-space search and problem solving
theory basic search: $tags$ common to this file questions

@1:blind:depthFirst $alg$:Complexity: difficulty: simple

:time

@2:guided:aStar type: single

Algorithm $A^*$ is:
+ complete and reachable
+ informed
- guided
- heuristic
- nor complete nor reachable
- not complete but reachable
- complete but not reachable
- blind

Space complexity of depth-first search is:
+ $O(bm)$, where $b$ is the branching factor and $m$ is maximum tree depth
+ $O(b^m)$, where $b$ is the branching factor and $m$ is maximum tree depth
- identical to its time complexity
- constant
- polynomial
- $O(bd)$, where $b$ is the branching factor and $d$ is the depth of solution
- $O(d)$, where $d$ is the depth of solution
- $O(b^{d/2})$, where $b$ is the branching factor and $d$ is the depth of solution
- none of the above

Figure 1. An excerpt from a questions database.

In Fig.1, the user-specific tags are shown in bold. To make tagging less tedious, the tags in the file may be specified at three different levels, each of progressively narrower scope. At the top-most level, tags that are common to all questions in the file are specified (line 4). At the second level, each particular question is tagged (lines 6 and 36). Finally, at the third level, each particular question variant may be tagged (lines 8 and 22). Hierarchical relationship between tags is indicated with a colon. For example, tags search, blind, and depthFirst may be written as search:blind:depthFirst to reflect the fact that blind search is a kind of search procedure and that depth first search is in turn a kind of blind search. To make hierarchical tagging more convenient, we allow hierarchical tags to be broken down into parts and specified incrementally. This is accomplished by propagating tag specifications to lower scopes: a tag ending with a colon will be propagated to the immediate lower scope, while tags

```
starting with a colon will be appended to the tag that is being propagated. In example from
Fig. 1, tag search, being specified at the top-most level, is propagated to both questions @1
and @2, while tag algComplexity is propagated only to the two variants of question @1.
Thus, both variants of question @1, besides being tagged with theory, basic, and
difficulty:simple, will also be tagged with search:blind:depthFirst. In addition
to that, first variant of question @1 will be tagged with algComplexity:time, while the
second variant will be tagged with algComplexity:space. By the same token, question
@2, besides being tagged with theory, basic and type:single, will also be tagged with
search:guided:aStar. Question @2 is tagged with type:single to indicate that its
answers are somewhat overlapping and hence this question should not be used for tests in
which more than one correct answer is possible (multiple-select questions).

Because Enthusiast generates test sheets in LaTeX, it is possible to directly use LaTeX
formatting tags in both the question text and the answer options. This makes typesetting
of mathematical expressions especially convenient. In Fig. 1, the mathematical expressions
appear enclosed in LaTeX tags ‘$’. Associating images to questions is also straightforward,
but will not be demonstrated here.

2.2 Test Specification

The tag metadata provided in the questions database effectively defines user-specific
questions taxonomy. Based on this taxonomy, the user can provide Enthusiast with a detailed
test specification regarding the content and the structure of the test. For example, the user may
specify that the test should consist of six one-out-of-four questions, of which four are related
to the today’s lecture, two to the specific topic of the last week’s lecture, and of which one
should be more difficult than the other five. Based on this test specification, for each test sheet
Enthusiast will choose six appropriate questions from the database, as well as one correct and
three incorrect answer options for each of them. To even further improve the variability across
test sheets, the presentation order of questions and answer options may also be shuffled. On
the other hand, if required, one can specify that certain questions or answers should be
common to all test sheets, or that their presentation order should be fixed as well. This way,
the user is given full control over the content and variability of the test, ensuring a fair
assessment of students’ knowledge.

An example of a simple test specification file is given in Fig. 2. This specification is for an
end-of-lecture quiz with six one-out-of-four questions. In line 5, the maximum number of
correct answer options, number of total answer options, and number of questions is given
(note that, although Enthusiast can generate multiple-select tests, this type of test has been

| 1: | # Course: Artificial intelligence |
| 2: | # Test: Quiz 1 |
| 3: | |
| 4: | AI_2008 01 % course ID, test ID |
| 5: | 1 4 6 % max correct, num options, num questions |
| 6: | 7: theory basic -disclosed % tags common to this test |
| 8: | |
| 9: | 1 introduction |
| 10: | 2! search:blind |
| 11: | 3 search difficulty:simple |
| 12: | 4 aStar | -difficulty:simple |
| 13: | 5 complexity:space ! |

Figure 2. Example of a test specification.
argued against in [5]). In line 7 we specify that this quiz contains questions tagged with
basic and theory, but not tagged with disclosed (say we decided to use this tag for
questions that are already known to the students). Then follows a more detailed specification
for five out of six questions, leaving one question fully unspecified. The topic and type of
each question is defined by specifying which tags this question should (or should not) have.
For example, we specify in line 11 that question 3 should be about state-space search (tag
search) and rather simple (tag difficulty:simple). We can also build logical expressions
to express more complex specifications. For example, in line 12 we specify that question 4
should have tag aStar or not have tag difficulty:simple. Note that mutual exclusivity
of question variants is automatically enforced by Enthusiast.

Based on a test specification, Enthusiast will choose at random five suitable questions from
the questions database. Because nothing is specified for the sixth question, Enthusiast is free
to make a random choice among all questions in the database, provided these are tagged with
basic and theory and not tagged with disclosed. Unless question number is marked with
an ‘!’ , the presentation order will also be randomized. In example from Fig. 2, question 2 will
always appear second, whereas the presentation order of other questions will vary from sheet
to sheet. To constrain the variability of a particular question, one can add a ‘!’ at the end of a
question specification, as we did with question 5 in Fig. 2. This has the effect of Enthusiast
not varying the question among the test sheets. Thus, once Enthusiast has chosen a question
that is tagged with complexity:space, the one and the same question (or one of its
variants) will appear on each test sheet. We could have suppressed variability even further by
typing ‘!!!’ instead of ‘!’ , which would settle on a question variant, or even by typing ‘!!!!’,
which would additionally settle on the answer options.

2.3 Test Generation

Using the questions database and a test specification, Enthusiast generates automatically a
required number of paper-and-pencil test sheets (this number is given as command-line
argument). Questions that meet the test specification constraints, and the corresponding
answer options, are chosen at random from the questions database. However, if Enthusiast
cannot meet these constraints, it will complain to the user and ask him or her to revise the test
specification. This typically happens if a question with specified tags does not exist in the
database, but it can also be that the specification is simply over-constrained and thus
unsatisfiable. When choosing among questions from the database, Enthusiast will ignore and
warn about questions that are erroneous (e.g., have two identical answer options) or
inadequate (e.g., do not contain a minimal number of correct and incorrect answer options for
a given test type).

Because questions and answer options are chosen and ordered at random, test sheets will
differ among themselves to the extent allowed by the test specification and the size of the
questions database. To give the user a sense of that variability, upon generating the test sheets
Enthusiast will compute and report the mean number of overlapping questions between two
test sheets. Based on this feedback, the user can decide whether he or she wishes to improve
test variability by lessening the test specification constraints or by adding a few more
questions variants to the database.

The format of the test sheets is determined by a customizable LaTeX template. The template
defines the typographic appearance of the test sheet, such as positioning of questions on the
sheet, font, title, as well as additional graphic elements such as test sheet bar-code, etc. User
can change this template to suit his or her needs. In this way, the user can use full power of LaTeX to produce not only functional but also aesthetically pleasing output.

After generating the test sheets, Enthusiast produces two files. First file is a LaTeX document containing the specified number of test sheets. Using LaTeX, this document can be compiled into a high-quality Post-Script or PDF format. In Fig. 3 we give an example of a single test sheet generated using questions database from Fig. 1 and test specification from Fig. 2. This test sheet features a computer-readable answer form for automatic grading, a point we discuss below. Second file output by Enthusiast is a list of correct answers for each test sheet.

Figure 3. An example of a test sheet generated by Enthusiast.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>2.</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>3.</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>4.</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>5.</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>6.</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>
3 Practical Experience

*Enthusiast* was recently used in three computer science courses offered by our faculty: Computer graphics, Scripting languages, and Artificial intelligence. For the latter it was used to generate test sheets for both summative (mid-term and final exams) and formative (end-of-lecture quizzes) knowledge assessment. Although multi-choice tests were used in both cases, the tests obviously had to differ substantially in both form and content. In what follows, we present our practical experiences in using *Enthusiast* for course on Artificial Intelligence.

3.1 Summative Testing

For summative testing, we used *Enthusiast* to generate four to eight distinct test sheets, each containing 20 questions for mid-term exams and 25 questions for final exams. Each question provided six answer options. We used one and the same questions database for each test. This database totals over 200 questions and over 350 question variants, and is well augmented with metadata tags. Among others, in this database we distinguish between theory questions (tagged with *theory*) and problem questions (tagged with *problem*). For each exam, we wrote a test specification that ensures a good coverage of the course material, but also ensures that the majority of the questions are problem questions. The presentation order of questions, the presentation order of answers, and the answer options themselves were allowed to vary among sheets. However, in order to ensure fair assessment of students' knowledge, we decided to constrain somewhat the variability of questions themselves. In our view, for summative testing it is important that tests have identical questions, though it might be acceptable or even desirable that tests differ in question variants. As explained in Section 2.2, such constraints can easily be enforced by making *Enthusiast* settle on questions or question variants across all test sheets. In order to minimize test cheating, we allowed for question variants, but took care that variants differ only slightly. Following the recommendations from [6], we decided to penalize for wrong answer in order to prevent blind guessing.

One obvious advantage of multiple-choice tests is that they can be scored fast, especially if machine-scored. To support machine-scoring of tests, students marked their answers on a separate computer-readable answer sheet with student ID number encoded in bar-code. Based on the answers file generated by *Enthusiast*, the sheets were machine-scored and results were usually announced within three hours.

3.2 Formative Testing

For formative testing, we used *Enthusiast* to generate end-of-lecture quizzes with six basic comprehension questions, each with four answer options (see Fig. 3). We used the same questions database as in the above case, but had *Enthusiast* choose only the very basic theory questions (those tagged with both *theory* and *basic*), and of course only those related to the particular lecture. For each quiz, we wrote a test specification that ensured that questions are well balanced among lecture topics (see Fig. 2). Because we wanted the end-of-lecture quizzes to contribute to the final score, and because students took them in a full classroom (over 90 students took the course), preventing test cheating became a major concern. Thus, to minimize the chance of cheating, we generated a different test sheet for each individual student. The test sheets differed in questions and their presentation order, as well as the answer options and their presentation order. Each quiz used
on average 20 questions from the questions database. Even with such moderately-sized
questions database, for each quiz we managed to generate more than 90 test sheets with an
average of less than three overlapping questions between two sheets.

Rapid feedback to students is even more important for formative than for summative tests.
While machine-scoring of separate answer-sheets works well for summative tests, it is not
feasible for end-of-lecture quizzes because distributing individual answer sheets to the
students would take up far too much time. Instead, our approach was to combine together the
answer and the test sheet. Test sheets had an answer form printed on them, and the students
were asked to put on the sheets their own bar-code stickers (given to them at the beginning of
the semester). Moreover, the LaTeX template for test sheets was modified so that each test
sheet featured a unique bar-code identifier (see Fig. 3). Machine-scoring of tests then paired
the student identifier with sheet identifier, and read off the answers that the student had
provided on the same sheet. The results were usually announced within an hour's time,
thereby providing rapid feedback to students. Seven end-of-lecture quizzes were administered
during the semester, with positive comments from students.

4 Related Work

There exists a number of multiple-choice test generation software, such as Question Mark [7],
ExamGen [8], HotPotatoes [9], and TestPilot [10]. The latter two generate web- or computer-
based tests and cannot actually be used to produce paper-and-pencil tests. Question Mark, on
the other hand, is a full-blown commercial product for authoring, scheduling, delivering, and
reporting on tests. Although its functionality extends far beyond that of Enthusiast, Question
Mark is not really meant for paper-and-pencil testing and seems to lack some peculiar
features, such as the ability to control variation across test sheets or define mutually exclusive
question variants.

Most similar to Enthusiast is ExamGen, a GUI-based Java application that can be used to both
manage multiple-choice questions (stored in a Microsoft Access database) and generate
printable test sheets (in HTML format). Besides multiple-choice questions, the user can define
short-answer questions, the inclusion of which, however, prevents full machine-scoring of the
test. A useful feature of ExamGen, one that is missing in Enthusiast, is the ability to keep
track of when a particular question was last used in an exam. On the other hand, ExamGen is
missing a number of important features, notably the ability to generate randomized test sheets
and the possibility to provide a redundant number of correct and incorrect answer options.
Based on our practical experience, we consider these features to be absolutely necessary for
large class assessments. Moreover, while in Enthusiast one can build elaborate questions
taxonomy with respect to both the topic and type of questions, in ExamGen one can merely
group questions according to user-defined categories. The possibility to build taxonomies of
questions and to refer to these in test specifications is important as it supports the use of one
and the same question database not only for different tests, but also for different kinds of tests.

5 Conclusion and Future Work

Enthusiast is a flexible tool for the automatic generation of pen-and-pencil multiple-choice
test sheets. It uses a plain-text database of multiple-choice questions and a test specification
provided by the user to generate randomized multiple-choice test sheets suitable for machine-
scoring. In order to improve the variability across test sheets, each question in the database
may be complemented with mutually exclusive question variants, as well as a redundant
number of correct and incorrect answers. The questions database may be augmented with
metadata tags, effectively defining user-specific questions taxonomy. The tags themselves may be organized hierarchically, allowing for more fine-grained and more comprehensive taxonomy. Based on this taxonomy, a test specification can be written that gives the user full control over the content and type of the test, and the variability across test sheets.

Enthusiast has been used in several courses at our faculty, for both summative and formative knowledge assessment. Based on our experience, we are confident that Enthusiast can be used to generate test sheets of adequate variability, provide for significant time savings, and ensure rapid feedback to students.

As part of future work, we intend to develop a web-based interface to Enthusiast that integrates test authoring, test sheets generation, and automatic grading. If the need arises, we will consider how to extend test specification format to allow for more flexible specifications. We also intend to extend the tagging system to keep track of when a particular question was used in an exam.

Acknowledgments

This work has been supported by the Ministry of Science, Education and Sports, Republic of Croatia under the grant 036-1300646-1986.

References:


Author(s):

Jan Šnajder, mr. sc.
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, 10000 Zagreb, Croatia
Jan.Snajder@fer.hr
Marko Čupić, mr. sc.
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, 10000 Zagreb, Croatia
Marko.Cupic@fer.hr

Bojana Dalbelo Bašić, prof. dr. sc.
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, 10000 Zagreb, Croatia
Bojana.Dalbelo@fer.hr

Saša Petrović
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, 10000 Zagreb, Croatia
Sasa.Petrovic@fer.hr