Five Years with Kattis – Using an Automated Assessment System in Teaching

Emma Enström, Gunnar Kreitz, Fredrik Niemelä, Pehr Söderman and Viggo Kann
KTH – Royal Institute of Technology, emmaen@kth.se, gkreitz@kth.se, niemela@kth.se, pehrs@kth.se, viggo@kth.se

Abstract – **Automated assessment systems have been employed in computer science (CS) courses at a number of different universities. Such systems are especially applicable in teaching algorithmic problem solving since they can automatically test if an algorithm has been correctly implemented, i.e., that it performs its specified function on a set of inputs. Being able to implement algorithms that work correctly is a crucial skill for CS students in their professional role, but it can be difficult to convey the importance of this in a classroom situation.**

Programming and problem solving education supported by automated grading has been used since 2002 at our department. We study, using action research methodology, different strategies for deploying automated assessment systems in CS courses. Towards this end, we have developed an automated assessment system and both introduced it into existing courses and constructed new courses structured around it. Our primary data sources for evaluation consists of course evaluations, statistics on students' submitted solutions, and experience teaching the courses.

Authors of this paper have been participating in teaching all of the courses mentioned here.

Index Terms - Computer Science Education, Algorithms, Automated assessment, Programming.

RELATED WORK

The use of automated assessment systems worldwide is described in various sources [1, 2, 3, 4, 5]. Many teachers have tried ways of promoting scaffolded learning, or helping the students with decompositions of a greater task into subtasks [6, 7]. Rosenbloom has used contest-influenced tasks in an introductory course in programming [8]. Gárcia-Mateos and Fernández-Alemán describe the experience of reworking assessment and grading towards continuous examination and assessment combined with an automated assessment system [9]. Not surprisingly, this increased student activity and performance.

OUR CONTRIBUTION

Automated assessment systems clearly assist in reducing the teacher's workload by removing the tedious work of manually verifying correctness. While this motivates their usage in itself, we are interested in new possibilities arising in the space of potential teaching strategies, as exemplified in our discussion on reductions below. A final written exam is generally not the best way of demonstrating knowledge or skills in programming, or possibly in any subject [10]. Our programming exercises can serve both to practice programming and to illustrate theory [11], and to assess and grade, during the course and in the end of a course. Since 2005, we have used the automated tool Kattis, developed by us, to assess programming exercises. Many other systems with similar functionality exist [12, 13]. Kattis was developed iteratively with requirements from courses and evaluations from students as input. The system has shown to be flexible enough to be used in different courses with different didactic framings.

Kattis is also used for programming competitions, the most well-known and prestigious being the ACM International Collegiate Programming Contest (ICPC) World Finals. The main use of Kattis is in our advanced programming and algorithm courses for CS students, as tasks in these courses can be complicated and therefore difficult to assess accurately, objectively, and efficiently for teachers and teaching assistants (TAs). We suggest that using the system allows the teacher to take a new set of roles in the classroom, and also creates a more consistent, non-negotiable, message to the students about the requirements for the exercise.

PROGRAMMING EXERCISES AT OUR SCHOOL

Programming exercises are the core of the examination in our courses focusing on programming, but they also constitute an important part of other courses, such as algorithms or cryptography courses. The main purpose of an exercise can be either to practice implementing algorithms from specifications or pseudo code, or solving some problem by constructing a program. Another purpose is to make the students work continuously with the courses. The assessment criteria may, aside from correctness, also include efficiency in the form of time limits or other limitations of resources. The students usually work in pairs during scheduled computer lab hours and on their own time. During lab hours teachers and TAs are available for questions and help, as well as for assessing completed exercises. Many exercises are introduced by some preparatory questions in order to support the students in choosing appropriate data structures and algorithms. For each exercise there is some kind of deadline, often connected with bonus points for the exam. The exercises are often mandatory in order to get a passing grade, and sometimes the grade is heavily based on these exercises. In order to get an exercise accepted, the students have to present their solution to a teacher. The traditional way of doing this is that the students, at a scheduled computer lab hour, show their program to the teacher/TA, who checks that the code meets the standards relevant for the course, that the students can explain what they have been doing and how their program works and, last but not least, that the program seems to work. This can be done for instance with a list of suitable inputs for testing, that the teacher can sample from. When time limits exist, the teacher must also check that the program is fast enough.

The presentation not only constitutes an examination situation – the TAs try to challenge the students and give them relevant feedback for future work as well as check that students understand the problem they have solved and can argue for their chosen solution. Both tasks are important, but require the TA to function both as the barrier students have to pass to finish the assignment and as a coach. Time used by the TA to test and inspect the program for flaws or bugs, is valuable teaching time lost. Additionally, if the program is found not to function satisfactorily, the TA also has to disappoint the students by failing them. Hence, less time is spent on discussing more interesting aspects of problem solving or programming for the exercise. If the teacher ends up only testing correctness, failing students and arguing why this is necessary, everyone will feel miserable afterwards.

To solve this problem, an automated assessment system was constructed – Kattis.

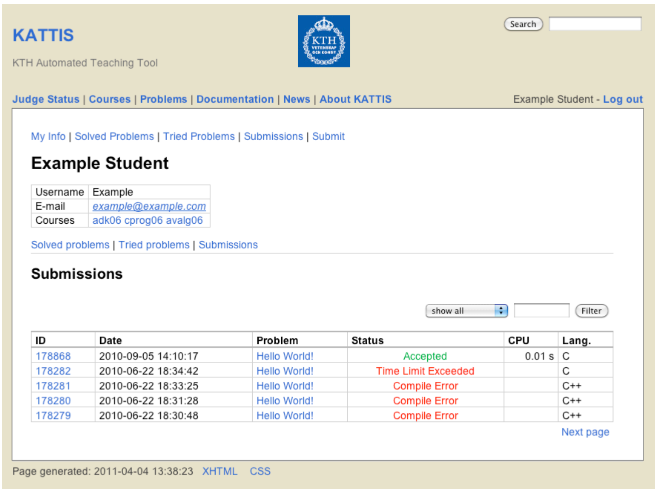
SYSTEM DESIGN

Kattis is a client-server system accessed via web and e-mail interfaces. The system is available to the students at all times, and not just during lab hours. There are four fundamental objects in the Kattis system. These are the user, the problem, the submission and the assessment. A user is allowed to create submissions on a problem. Once a submission has been made the judge system will create a "judgement" for the submission and report this back to the user. The design is based on ACM ICPC style competition systems, and the terminology used in the documentation also originates from programming competitions. Hence Kattis is a "judge" and "judges" solutions to problems. Originally the "verdicts" or "judgements" were minimal, only reporting the outcome (such as "Accepted," "Wrong Answer," or "Run Time Error.") As in ICPC style competitions, students' solutions are tested using inputs which are kept secret from the students.

I. Web interface, E-mail and Command-line Interface

The web interface is the main interface to Kattis, see Fig.1. It provides historical data about all submissions and data about the current status of the Kattis system. All students have usernames and passwords, which they use to access the system and their private pages, which include the source code for all of their submissions. They can submit the source code of a program over the web. For each accepted submission, Kattis measures the CPU time used. The results of submissions are public, but a student can opt to hide her name.

Alternatively, an e-mail interface can be used. This allows students to e-mail the source code of a program directly to the system, providing authentication information in the e-mail. Kattis can also e-mail back the result of the submission to the student, providing feedback, in addition to showing it in the web interface. A command-line tool can also be used.



KATTIS

KTH Automated Teaching Tool

Judge Status | Courses | Problems | Documentation | News | About KATTIS

Example Student - Log out

My Info | Solved Problems | Tried Problems | Submissions | Submit

Example Student

Username	Example
E-mail	example@example.com
Courses	adk06 cprog06 avalg06

Solved problems | Tried problems | Submissions

Submissions

ID	Date	Problem	Status	CPU	Lang.
178868	2010-09-05 14:10:17	Hello World!	Accepted	0.01 s	C
178282	2010-06-22 18:34:42	Hello World!	Time Limit Exceeded		C
178281	2010-06-22 18:33:25	Hello World!	Compile Error		C++
178280	2010-06-22 18:31:28	Hello World!	Compile Error		C++
178279	2010-06-22 18:30:48	Hello World!	Compile Error		C++

Next page

Page generated: 2011-04-04 13:38:23 XHTML CSS

FIGURE 1
EXAMPLE OF WEB INTERFACE FOR STUDENTS.

II. Backend

The Kattis backend is built in Python, with minor parts written in C and Java that are used in the security solution. Data about users, submissions and assessments are stored in a PostgreSQL database. Metadata about problems are also stored in the database, while the input and output files for the problems are kept in the file system. When a submission is made, it is immediately stored in the database and the backend is informed that it should judge the submission. The backend then retrieves the submission from the database and stores the source code in a temporary directory. After this the backend compiles the source code, using any compiler flags and options specified for the problem. A submission can at this point fail with a "Compile error," if compilation fails. The next step is to run the submission. The program is wrapped with a security layer, which prevents potentially dangerous system calls, and then executed with the input file on standard input. It writes its answer to standard output.

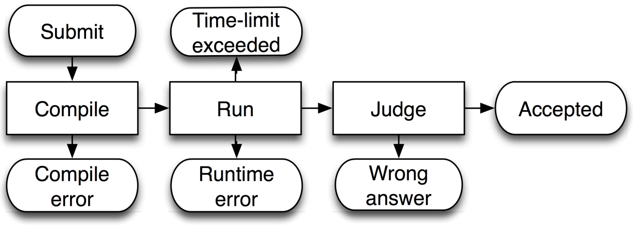


FIGURE 2
MOST COMMON PATHS OF PROBLEM FLOW IN KATTIS.

At this point the program can fail for one of several reasons, such as "Runtime error," "Time-limit exceeded," or in exceptional cases even "Illegal function." If the program completes the run successfully, the output is either compared to the reference output for the problem, or processed by a program that does something more complex such as verifying properties of a graph. This allows several correct answers. If the student's program gave an incorrect answer the submission is considered a "Wrong Answer," otherwise the judge checks if there are any more input files. If it is the last input file the submission is judged "Accepted," see Fig. 2.

METHODOLOGY FOR TEACHING WITH KATTIS

Kattis is designed to be used with what we call test-driven education. The aim of introducing Kattis was to change the process of getting a solution accepted to include more opportunities of verification and hopefully no surprises at presentation. Before presenting their program to the teacher, students are required to submit it to Kattis in order to test for correctness. They can submit solutions as often as they prefer and whenever they want. In this way, Kattis takes the role of an adversary, failing incorrect solutions. Working against an adversary encourages a test-driven development process for the student's solution. If the exercise is partitioned in appropriately sized pieces, this will provide some scaffolding for the learning process. The work that Kattis performs therefore has the character of formative assessment, but as soon as the student is satisfied with the result, the assessment becomes summative and sometimes decides the student's grade.

When presenting the program, students are asked to show that Kattis accepted it, which means that both the teacher and the students know that the program works at the beginning of the presentation. The rest of the presentation is as before; that is, students have to explain their code and choice of strategy and the teacher asks questions like "what does this parameter mean" or "what happens if you change..." The combination of manual and automated assessment has been previously studied and recommended [14]. By removing the tedious work of testing the program from the teacher, the importance of the rest of the presentation is emphasized. We claim that this is beneficial to all parties involved: the student, teacher, and the university. We can provide more exercises at a lower cost, and the quality of teacher-student meeting time increases. For the student, already knowing that the program works when signing up to present the work, leads to a less stressful situation.

Kattis is guaranteed to treat all students equally, and chance does not affect whether a submitted program is accepted or not. All test cases are always used. Hence, Kattis assesses the solutions' correctness better that the teacher would.

EXPERIENCE

Kattis was first used in two courses in 2005 and 2006. After the first course, interviews were conducted with two students, and after the second all students got a few questions about the system. From this, we learned that the major drawbacks perceived by students were that it was frustrating knowing that the program worked for most inputs, but not exactly for what input it failed, and partly that for some exercises, the standard libraries for input and output handling in Java and C++ were too slow, so that the students needed to work not only on the algorithms they were supposed to practice on, but also on reading and writing. This was later remedied by providing example programs with methods that were sufficiently efficient.

Regarding the secret test cases, it was never our intention to guide the students with them. The solution should be based on the problem statement, as should the test cases. If provided with the exact test cases, students would face an entirely different, and simpler, task. However, when a program was failed by Kattis, a teacher interface with more information about each submission (including the failing test cases) was needed, and therefore built. The decision whether to tell the students what the test case looked like then belonged to the teachers. This makes it possible to help students who are genuinely stuck trying everything they can come up with, without rewarding non-reflective behavior among the students as a group.

The feedback mechanisms were also improved so that each test case can be run separately and a failure hence was associated with a specific test case and feedback. Statistics of solutions for the problems were also included, together with a high score list for each problem. After this, Kattis was tested on more courses.

CASE STUDIES

I. Algorithms, Data Structures, and Complexity

Algorithms, Data structures, and Complexity (ADK) is a second (from now on third) year course of about 130 students, mandatory for CS majors. Writing efficient programs is among the goals of the course. Four programming exercises constitute a third of the course. Since the course is on fairly complicated algorithms and about solving problems efficiently, the testing needs to be extensive and the teacher might fail to spot errors such as corner cases that are not handled correctly. There could also be an issue that certain computers are faster than others, so checking time limits could include running the program on some special machine. As hints for the problems in the exercises, there are theory questions that students can answer before they start programming. These questions are marked in class by peer assessment.

One of the exercises was split into several parts when Kattis was introduced, and another exercise was added after Kattis' first appearance in the course.

II. Programming and Problem Solving Under Pressure

Programming and Problem Solving Under Pressure (Popup) is an elective advanced level course of about 30 students, most of them CS majors. This course was the first course where Kattis was used, and the course is designed around the concept of test-driven education. Part of the examination is for students to solve a very large set of problems; a total of at least 26 solved problems over a semester are required for a passing grade.

The problems used in the course are of the same style as those used at algorithmic programming competitions, i.e.

small (a solution typically has 25-100 lines of code) and well-defined. The formal input specification includes limits on how large the test cases are. Another part of the examination for the course is that students are asked to produce a programming library. By creating high-quality implementations of algorithms, and having them automatically tested for correctness by the Kattis system, students can be confident that they have implemented the algorithms correctly. This allows them to confidently use their library as part of solutions for other problems, both in this course, and in their future work. Without an automated grading system, the workload of grading literally thousands of student solutions would be overwhelming.

III. Advanced Algorithms

The Advanced Algorithms course has used Kattis since fall 2006, and needed more advanced feedback features such as detailed error messages and the ability to assign scores to solutions based on their quality, as specified by the teacher. The output is collected by a program that computes a score, for instance based on the number of test cases a factoring algorithm was able to finish within the time limit or the length of a tour in the Traveling Salesperson problem (TSP). This was essential, since students had begun competing for the best solution, and unless otherwise specified, the high score list will show the fastest solutions. The teacher then needed to direct the competing students' attention to the quality criteria of interest for the problem at hand. There were no changes in passing rate or grade when Kattis was introduced, but students got more feedback earlier and many of them continue to submit solutions long after their first one is accepted by Kattis.

Although our hypothesis is that more students strive to get further in terms of quality of their solutions, we cannot prove this, since several external factors have changed during our evaluation period. These include new grades that were introduced, varying admission grades, and changes in exercises over the years.

IV. IP routing in simple networks

IP routing is an advanced course in networking, given to master level CS and Telecommunication students. The course is practical in nature, with a focus on configuration of networks and detailed understanding of protocols. The number of students has been steadily increasing, from 16 in 2007 to 65 students in 2010.

A homework assignment to write a program that forwards IP packets was introduced in 2007. The assignment is intended to provide students with a deeper understanding of the practical issues of packet processing in a router, including the handling of endianness of data, unaligned data structures and lookups in large routing tables. This task was considered hard: 31% of the students failed to complete it in 2007 and 50% in 2008. The low performance was primarily attributed to lack of experience with the C programming language among the students. To improve student performance the assignment was converted to use the Kattis system in 2009. At the same time the assignment was made easier, anticipating increased difficulty from the more careful validation made by Kattis. An option was also introduced to allow students lacking experience with the C programming language to complete a theoretical task instead. The result was a considerable improvement in student performance (9% failed 2009 and 13% failed 2010). Around 12% of the students choose to complete the theoretical task instead of the Kattis assignment each year.

THE EFFECT OF CAREFULLY CHECKING STUDENT CODE

The first year Kattis was used in the ADK course, 2006, fewer students actually were accepted on time for one of the assignments. We have no reason to believe that students got into unnecessary trouble because of having to use Kattis, to the extent that they did not pass the examination. However, it is likely that the phenomenon occurred since the code that the students produced often was not correct or did not follow the specifications, neither before nor after the introduction of Kattis. Since all teachers cannot always run all test cases, it was possible before Kattis to pass the presentation with an *almost* correct program. With the Kattis system, a much more careful, and standardized test of the students' code is made. Table I shows the results over time in the ADK course, where the fraction of students who passed the assignment during the course is listed. Some differences in performance can seemingly be explained by the students' general performance level – for instance in 2005 all results dropped and 2007 the performance was better than before and after. For this reason, the admission grade and performance level of the students of their first year is listed for comparison. The maximum admission grade is 20.0 and the grades in the table are the lowest among the accepted students.

TABLE I
STUDENTS ACCEPTED WHEN PRESENTING EXERCISES IN ADK. E1 AND E2 WITH KATTIS FROM 2006 AND E3 WITH MANUAL ASSESSMENT ONLY.

year	E1	E2	E3	Admission grade (min.)	Done with ≥2/3 of first year courses
2000	90 %	94 %	94 %	18.28	-- %
2001	92 %	91 %	88 %	17.84	-- %
2002	89 %	90 %	88 %	17.55	-- %
2003	93 %	97 %	93 %	16.96	-- %
2004	94 %	92 %	88 %	15.33	76 %
2005	83 %	83 %	77 %	14.50	52 %
2006	71 %	83 %	84 %	16.18	55 %
2007	88 %	89 %	91 %	15.70	67 %
2008	74 %	83 %	78 %	11.52	66 %
2009	77 %	83 %	88 %	15.10	63 %
2010	80 %	85 %	85 %	15.43	64 %

Exercises E1 and E2 became more difficult to finish on time when Kattis was introduced. The three exercises listed have been re-ordered through the years, but apart from adding Kattis, no major changes have been made.

STUDENTS' MOTIVATION AND KATTIS USAGE

Two things in the web interface seemingly have especially interesting consequences: the fact that Kattis publishes the CPU time a submission consumes and that Kattis for each problem and programming language publishes a "high score list" listing the best solutions, by speed or by score.

The possibility of using Kattis outside regular working hours is not only a theoretical advantage – it is used extensively by the students, see Fig. 3. For the exercises that are used in the ADK course, students on average have between 2 and 4 accepted submissions, but the variation is large. The median number of accepted submissions for each problem is one or two, but more than 50 students have five or more, and some students have more than 100 accepted solutions to the same problem.

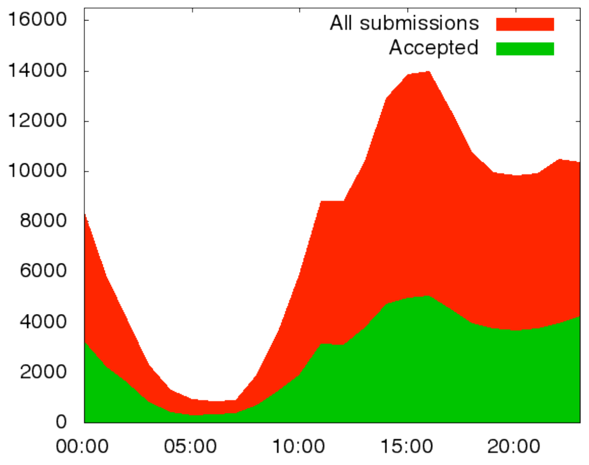


FIGURE 3
SUBMITTED AND ACCEPTED SOLUTIONS FOR DIFFERENT HOURS OF THE DAY.

COURSE EVALUATIONS

Each course is evaluated after the final exam, and some questions usually concern Kattis when Kattis has been used. The first year Kattis was used on the ADK course, only half of the students thought that Kattis was a good or a very good service. This number has since increased to over 80 %. The students are happy about the existence of Kattis, however, each year some of them want to see more feedback and information when their solutions are not accepted by the system. The actual test cases are also desired during debugging, but not given since that would change the comprehensive content of the exercises completely. The exercises would only be about programming according to a specification that someone else is responsible for.

DISCUSSION

Sometimes students complain about Kattis being unreasonable. Behind their reasoning, seemingly the idea resides that it is a two-step process to finish an assignment – first make it work, and then get it accepted by Kattis. This could be due to sparse feedback [13], but we believe in another explanation: while teachers believe that correct handling of *all* legal input is crucial, students seem to consider a program correct even if some corner cases are treated incorrectly, as found by [15]. By using Kattis, the teachers' view is enforced in the final programs. Students appear to more easily accept an automated system being pedantic.

Considering Kattis "game-like" raises new issues to take into account. When playing an educational game, it is not necessarily the case that students see the task from an educational perspective. They might instead set out trying to outperform themselves or other students, learning the game in detail – enter "Nintendo mode" [16]. In programming exercises, generally one of the goals is learning to program better, and hence merely working with the exercises is guaranteed to provide programming experience. For the majority of the exercises in Kattis, the notion of "Nintendo mode" is not a problem, since the "game" consists of tasks and requires competences that we want the students to learn. Students trying to write the fastest program are spending more time on the task, which is one condition for learning [17].

The time limits are not only used by teachers to enforce efficient solutions. They also motivate students to improve and to compete. Of course, students might do this without Kattis, but having a solution on the list of fastest solutions or highest score for a problem seems to provide extra motivation. This adds some artifacts of games to the tasks, but a more detailed analysis is beyond the scope of this paper.

SPACE OF NEW POSSIBILITIES

Apart from the workload issue, the use of an automated system allows the teacher to be, psychologically, more on the students' side. Instead of grading, and sometimes failing, students' solutions the teacher's or TA's role now becomes more of a helper, assisting the student to get her program to be accepted by the assessment system.

Moving the testing of correctness and efficiency to an automated system is by itself a sufficient reason to use Kattis in a course. But in fact several new options become available through the use of Kattis, enabling the pedagogy of a course to come closer to test-driven education. While there are many interesting directions, we limit our discussion to two types of improvements, which have been implemented.

I. Splitting an Existing Assignment

As an automated assessment system is always available to test students' solutions, splitting an assignment into smaller pieces becomes a possibility. A typical traditional CS assignment may involve implementing a number of data structures and algorithms, and then correctly applying these to solve the problem. As manual grading is a time-consuming activity, typically only the complete solution is then graded by a teacher. Such inspection of the finished work can often reveal bugs that can be very difficult to correctly diagnose, as they may stem from bugs in many different parts of the solution. Apart from the grading scenario, it is a difficult and time-consuming task for TAs to help students with subtle bugs in a core algorithm causing a complex student solution to crash, seemingly at random.

By splitting the problem into pieces, and in most programming tasks there are natural pieces, the correctness and efficiency of individual pieces can be established. In a way, this is similar to unit testing, which is considered a best practice in the software industry.

II. Reductions

A core concept in computer science is the concept of reductions, both as tools for proofs and for constructing algorithms. For many students this concept is difficult to grasp. Through an automatic system, a student can be given black box access to something that can solve instances of some specific problem. The student is then asked to solve some other problem through the use of the black box, thus getting first-hand experience at discovering and implementing a reduction. The problem is set up in such a way that the student cannot solve it within the specified time limits without using the black box. This is important, as some students are otherwise prone to misunderstand the concept of reductions [18, 19] and attempt to implement a solution from scratch.
We have created several lab assignments where the student's solution interacts in a black-box fashion with a computer program solving some different task. In the first type of reduction, the student is simply given a black-box implementation of an algorithm solving the maximum flow problem that they interact with to solve bipartite matching, and in the second one the students get input for a known NP complete problem and the task to reduce it to another decision problem [11] with the same answer "yes" or "no."

CONCLUSIONS

Using an automated assessment system for educational purposes helps teachers to get time for essential teaching, releases the burden of marking/assessing and determining whether code is correct and makes students feel more confident about their solutions. This improves the relationship between students and teachers during oral exams, and provides tools for the students to work more independently of teachers. Not only mechanical work can be performed and assessed within this type of teaching. Some tasks require a large degree of creativity and high-level understanding of the problem to solve. The system can either be "added" on top of an existing course, introduce new types of assignments, or be used as the basis for a completely new course with a different teaching style and requirements than most courses. The phenomenon that the system encourages competition and makes students try hard has to be taken into account when introducing new exercises. Since the students make use of the information that they get, the resources that the system measures ought to be important. Otherwise, the students might still try to get a better number for the measurements, forgetting other goals with the exercise.

REFERENCES

[1] Sant, J. A. "'Mailing it in': email-centric automated assessment." *ITiCSE '09: Proc. 14th ann. SIGCSE conf. on Innovation and technology in Comp. Sci. education*, ACM, 2009, pp. 308-312.

[2] Amelung, M., Forbrig, P. and Rösner, D. "Towards generic and flexible web services for e-assessment." *ITiCSE '08: Proc. 13th ann. SIGCSE conf. on Innovation and technology in Comp.Sci. education*, ACM, 2008, pp. 219-224.

[3] Higgins, C. A. and Bligh, B. "Formative computer based assessment in diagram based domains." *ITiCSE '06: Proc, 11th ann. SIGCSE conf. on Innovation and technology in Comp. Sci. education*, ACM, 2006, pp. 98-102.

[4] Suleman, H. "Automatic marking with Sakai." *SAICSIT '08: Proc. 2008 ann. research conf. of the South African Inst. of Comp. Sci. and Information Technologists on IT research in developing countries*, ACM, 2008, pp. 229-236.

[5] Thomas, P., Waugh, K. and Smith, N. "Generalised diagram revision tools with automatic marking." *ITiCSE '09: Proc.14th ann. SIGCSE conf. on Innovation and technology in Comp. Sci. education*, ACM, 2009, pp. 318-322.

[6] Sooriamurthi, R. "Introducing abstraction and decomposition to novice programmers." *ITiCSE '09: Proc.14th ann. SIGCSE conf. on Innovation and technology in Comp. Sci. education*, ACM, 2009, pp. 196-200.

[7] Ginat, D. "Interleaved pattern composition and scaffolded learning." *ITiCSE '09: Proc. 14th ann. SIGCSE conf. on Innovation and technology in Comp. Sci. education*, ACM, 2009, pp. 109-113.

[8] Rosenbloom, A. "Running a programming contest in an introductory computer science course." *ITiCSE '09: Proc. 14th ann. SIGCSE conf. on Innovation and technology in Comp. Sci. education*, ACM, 2009, pp. 347-347.

[9] Gárcia-Mateos, G. and Fernández-Alemán, J. L. "A course on algorithms and data structures using on-line judging." *ITiCSE '09: Proc. 14th ann. SIGCSE conf. on Innovation and technology in Comp. Sci. education*, ACM, 2009, pp. 45-49.

[10] Falchikov, N. *Improving Assessment Through Student Involvement*, Routledge, New York, 2005, pp. 32-58.

[11] Enström, E. and Kann, V. "Computer lab work on theory." *ITiCSE '10: Proc. 15th ann. Conf. on Innovation and technology in Comp. Sci. education*, ACM, 2010, pp. 93-97.

[12] Leal, J. P. and Silva, F. "Mooshak: a web-based multi-site programming contest system." *Softw. Pract. Exper.,* Vol. 33, No 6, 2003, pp. 567-581.

[13] Ihantola, P. et al. "Review of Recent Systems for Automatic Assessment of Programming Assignments" *Koli Calling 2010, Koli, Finland*

[14] Ahoniemi, T. and Karavirta, V. "Analyzing the use of a rubric-based grading tool." *ITiCSE '09: Proc. 14th ann. SIGCSE conf. on Innovation and technology in Comp. Sci. education*, ACM, 2009, pp. 333-337.

[15] Kolikant and Ben-David, Y and Mussai, M. "So my program doesn't run! Definition, origins, and practical expressions of students' (mis)conceptions of correctness." *Computer Science Education*, Vol. 18, No. 2, 2008, pp. 135-151.

[16] Rieber, L. and Noah, D. "Games, simulations, and visual metaphors in education: antagonism between enjoyment and learning." *Educational Media International*, Vol. 45, No 2, June 2008, pp.77-92.

[17] Chickering, A.W. and Gamson, Z.F. "Seven principles for good practice in undergraduate education" *American Association of Higher Education Bulletin*, vol.39 no.7, 1987, pp.3-7.

[18] Armoni, M. "Reductive thinking in a quantitative perspective: the case of the algorithm course." *ITiCSE '08: Proc.13th ann. conf. on Innovation and technology in Comp. Sci. education*, ACM, 2008, pp. 53-57.

[19] Armoni, M., Gal-Ezer, J. and Hazzan, O. "Reductive thinking in undergraduate CS courses." *ITiCSE '06: Proc. 11th ann. SIGCSE conf. on Innovation and technology in Comp. Sci. education*, ACM, 2006 pp. 133-137.