Review of Recent Systems for Automatic Assessment of Programming Assignments

Petri Ihantola
Department of Computer Science and Engineering
Aalto University
petri@cs.hut.fi

Tuukka Ahoniemi
Digia Plc
Finland
tuukka.ahoniemi@digia.com

Ville Karavirta
Department of Computer Science and Engineering
Aalto University
vkaravir@cs.hut.fi

Otto Seppälä
Department of Computer Science and Engineering
Aalto University
oseppala@cs.hut.fi

ABSTRACT

This paper presents a systematic literature review of the recent (2006–2010) development of automatic assessment tools for programming exercises. We discuss the major features that the tools support and the different approaches they are using both from the pedagogical and the technical point of view. Examples of these features are ways for the teacher to define tests, resubmission policies, security issues, and so forth. We have also identified a list of novel features, like assessing web software, that are likely to get more research attention in the future. As a conclusion, we state that too many new systems are developed, but also acknowledge the current reasons for the phenomenon. As one solution we encourage opening up the existing systems and joining efforts on developing those further. Selected systems from our survey are briefly described in Appendix A.

1. INTRODUCTION

Assessment provides the teacher with a feedback channel that shows how learning goals are being met. It also ensures for an outside observer that students achieve those learning goals. Assessment provides both means to guide student learning and feedback for both the learner and the teacher about the learning process – from the level of a whole course down to a single student on some specific topic being assessed.

Students often direct their efforts based on what is assessed and how it affects the final course grade [6, Chapter 9]. Continuous assessment during a programming course ensures that students get enough practice as well as get feedback on the quality of their solutions. Providing quality assessment manually for even a small class means that feedback can not be as instant as in one-to-one tutoring. When the class size grows, the amount of assessed work has to be cut down or rationalized in some other way. Automatic assessment (AA), however, allows instant feedback without the need to reduce exercises.

Why do so many automatic assessment systems exist, and why are new ones created every year? Many systems share common features and it would seem that systems already exist that fulfill most assessment needs.

One clear reason for the variety of tools has to do with their availability and lifespan. Tools are often created as a part of a thesis or for a particular course. They are finished enough for studying a research question or to support the needs of one particular course, but are not suitable for distribution. It is rather common that the very first version of a tool was something that the teacher did quickly for his/her very own purpose. These tools might get publicized if some research was the original motivator, but as they never emerge as supported pieces of software, similar systems get implemented again and again. Correspondingly, there are far less systems that are widely adopted than there are papers about new tools.

We argue that presenting *a big picture* about the recently developed and currently available AA systems would help both teachers find the tools they might be searching and developers avoid reinventing the wheel. Literature survey is one way to achieve this. In this survey, our goal is to serve teachers who need to give grades to large classes. This is where the automatic grading of programming assignments can free the teachers' time significantly for doing something else, that can not be automated [9].

Related research, with focus on related surveys, is presented in Section 2. The exact research questions and the methodology used in this survey are described in Section 3. Results are introduced in Section 4. Selection of AA systems, also mentioned in Section 4, are presented in Appendix A. Conclusions, some recommendations based on the data, and our expectations related to the future trends in automatic assessment of programming assignments are discussed in Section 5.

Koli Calling '10, October 28-31, 2010, Koli, Finland

2. RELATED WORK

Tools are an actively researched approach to support teaching programming. For example, a survey by David Valentine found that 18% of the papers published in SIGCSE conference between 1983 and 1993 were tools papers, whereas between 1994 and 2003 the number was 24.6% [78]. In the *Survey of Literature on the Teaching of Introductory Programming* by Pears et al. [55] from 2007, tools were the single largest group among papers classified between tools, curricula, pedagogy, and programming languages. Analysis by Sheard et al. [68] from 2009 also supports the importance of both assessment and tools. Top three themes in their classification of CS education research papers were: 1) ability / aptitude / understanding (40%), 2) teaching / learning / assessment techniques (35%), and 3) teaching / learning / assessment tools (9%).

Pears et al. have also summarized that tools that support teaching programming can be divided into:

- visualization tools (e.g. ANIMAL [60], Jeliot [49], and Tango [73]),
- automated assessment tools (e.g. TRAKLA2 [40], WEB-CAT [16], and BOSS [37]),
- programming support tools (e.g. BlueJ [5]), and
- microworlds (e.g. Karel [54], and Alice [12]).

As stated before, *our focus is on tools for automated assessment of programming assignments.*

There are a few surveys of AA in the context of programming assignments. *A Survey of Automated Assessment Approaches for Programming Assignments* [1] by Kirsti Ala-Mutka from 2005 concentrates on what features of programming assignments are automatically assessed whereas Douce et al. [14] review the history of the field from 1960s to 2005. One of the main findings by Ala-Mutka is that dynamic analysis – that is, assessment based on executing the program – is often used to assess functionality, efficiency, and testing skills. Static checks that analyze the program without executing it are used to provide feedback from style, programming errors, software metrics, and even design. Tools that cover both static and dynamic testing are also well presented in the survey. There are many features to assess, and Ala-Mutka concludes that the selected AA approach should always be pedagogically justified. Although we believe a lot has been done since 2005, a recent survey from 2009 by Liang et al. [42] provides little new to the work of Ala-Mutka.

ITiCSE 2003 working group led by Carter conducted a survey among CS educators (not only programming) to get a snapshot of AA practices and an analysis of respondents' perceptions of automatic assessment [9]. One interesting finding was that the teachers who were not familiar with AA considered its potential more limited than the respondents with experience from AA.

Not all programming exercises can be automatically assessed. Several articles discussed how to design good assignments from the pedagogical standpoint. However, how to deal with the restrictions set by the automatic assessment is not often addressed. Forišek investigated International Olympiads in Informatics[1] (an event similar to the ACM International Collegiate Programming Contests[2]) and found that certain types of assignments they used were unsuitable for automatic assessment [17]. Forišek presents concrete examples of bad assignments (i.e easy to cheat tests) and heuristics on how to detect them. Greening, on the other hand, suggests that programming assignments should be more open in nature instead of satisfying a strict set of specifications often required by automatic assessment [25]. Furthermore, we believe that the very fact that the assessment is automatic is likely to change how some students approach the exercise. Knowingly submitting a weak or even incorrect solution that gets accepted by a machine is quite likely more socially acceptable than trying to cheat a person.

Some of the research outside CS education research will also help us understand when to apply AA and when not to. For example, what kind of problems in code can be detected automatically and what not has been investigated (e.g. [45]).

[1] `http://ioinformatics.org/`

[2] `http://cm.baylor.edu/`

3. RESEARCH QUESTION AND METHOD

Based on the previous section, we conclude that the trends and improvements in automatic assessment of programming assignments from the last five years have not been systematically collected. Thus, the following research questions are addressed in this paper:

1. What are the features of automatic assessment systems reported in the literature after 2005[3]?
2. What future directions are indicated?

To answer the questions, a systematic literature review was carried out. This means that an explicit procedure in selecting the systems and papers was applied (see [7] for details): to be included, a paper must have presented an AA system providing *summative*, *numerical* feedback from *programming assignments* or described results from using such system.

By programming assignments we mean assignments where students write code and submit it for assessment. Therefore AA of diagrams (e.g. [76]), AA of algorithm simulation (e.g. [40]), and other visualization based approaches (e.g. providing formative feedback based on visualizations) are not included. In addition, we only included systems where first hand experience was reported. This means that classical systems often mentioned in the related work section (e.g. CourseMarker [30], Assyst [34], etc.) are left out – unless experiences from those systems were reported in the literature we surveyed.

We collected the data by searching for phrases ('automatic' OR 'automated) AND ('assessment' OR 'grading') AND 'programming' from the conference proceedings and journals through *ACM Digital Library* and *IEEE Xplore*. We then applied the inclusion criteria to the abstracts and finally read all the remaining papers. Search terms were collectively decided after first manually examining (reading the abstract and scanning the rest) all the articles of ITiCSE proceedings and three journals: Computer Science Education, Olympiads in Informatics, and Transactions on Computing Education (formerly Journal on Educational Resources in Computing) between 2006 and 2010.

Because of our inclusion criteria, not all of the systems included in this survey are first published after 2005. A

[3] 2005 is when the survey of Ala-Mutka was carried out.

system might have been published earlier but an evaluation study or something similar after 2005.

We applied an iterative process to find the consensus about how to group features of the systems (i.e. subsections of Section 4). We read a selection of papers, made the first version of categories, read more papers and revised the categories. This was repeated until no significant features leading into new categories were found. Some of our results are explained by our background in automated assessment. As a short summary, our attitudes towards automation are positive, we have all used AA several years in our courses, and we have been developing AA and other educational systems.

4. RESULTS

In this section, we introduce the features identified in the literature survey. Systems are cited, but the focus is on features, not on systems.

4.1 Programming Languages

A majority of the systems are either targeted only for Java or have support for Java. This fits well with the trend of Java being one of the most used introductory programming languages. Other popular languages supported by the systems include C/C++, Python, and Pascal. Pascal was especially used in most of the competition platforms, where a typical language support also included C/C++ and Java. In addition, we found examples of not so main stream teaching languages like Assembler [36, 43] and shell scripts [70].

Some of the systems are language independent. Especially if the assessment is based on output comparison, any language that can be executed on the same environment can be automatically assessed after the system is configured to compile and execute solutions in that language.

4.2 Learning Management Systems

Extending the existing learning management systems (LMS) like Moodle[4] to better fit into the special needs of CS education seems to draw increasing interest (e.g. [56, 62, 63], the forthcoming ITiCSE'10 working group, and many others). One argument supporting LMS–AA integration is to avoid reimplementing all the course management features. As a LMS hosting several (not only programming) courses has a huge number of users, it is a tempting target for attackers. Malicious code executed in such an environment is always a serious threat. Therefore, securing AA systems integrated into LMSs is extremely important. Despite the challenges, we believe that there are more pros than cons in this approach and that there is an increasing demand to bring automatically assessed exercises into LMSs.

We found the following AA extensions to LMSs: CTPracticals [27] to bring VHDL and Matlab exercises into Moodle, Automatic Grader [74] to assess students' Java assignments in Sakai[5], AutoGrader [29, 51] to support Java assignments in Cascade[6], WeBWorK-JAG [21, 22, 23] to bring automatically assessed Java exercises into WeBWorK[7], SISA-EMU [36] to provide Assembler programming assignments through Moodle, and finally VERKKOKE [2] to bring socket programming and routing into any LMS with SCORM support (e.g. Moodle). In addition, EduComponents [3, 4] brings programming exercises to Plone[8], a content management system, not a fully featured LMS.

4.3 Defining Tests

Assessing the functionality of students' code is still the most often used approach to grade programs. The ways to do this can be divided between the use of industrial testing tools and various specialized solutions. Examples of using industrial tools were:

- **XUnit** based approaches were used in several systems (e.g. [3, 72]). In some cases students even created their own tests with JUnit [15].
- **Acceptance testing frameworks**, (e.g. EasyAccept [66, 67]) where tests are defined in a natural-language-like scripting language. Tests are easy-to-read requirement specifications as well as used for the assessment at the same time.
- **Webtesting frameworks** like Watir[9] in AWAT [75] and Selenium[10] in Electronic Commerce Virtual Laboratory [11].

Specialized solutions included:

- **Output comparison** is the traditional approach used by many of the systems we found. Survey of Ala-Mutka [1] already reported several variations of output comparison including running the model solution and student's code side by side and the use of regular expressions to match the output.
- **Scripting** can mean almost everything, and at the same time it is the most commonly reported way to define tests. For example, a script can be a shell script compiling the program, running it, and comparing the output to a file containing the expected output.
- **Experimental approaches** like comparing program graphs of a student's solution to the pool of known correct answers [50, 80] or deriving test cases with a model checker [33] were also reported.

4.4 Resubmissions

Practice is important in learning programming and there should be room for mistakes and learning from them. AA can help as it can give feedback despite the limited human resources. However, to prevent mindless trial-and-error problem solving, the number of resubmissions should be controlled [44]. Here are some examples of how the problem of trial-and-error can be tackled.

- **Limiting the number of submissions**, in addition to having deadlines, is the trivial approach supported by most of the current systems.
- **Limiting the amount of feedback** is another classical way to force students think after a failed submission. However, this can also create confusion among students. Especially, students not familiar with AA (who do not trust AA yet) may feel that the feedback

[4] http://moodle.org/
[5] http://sakaiproject.org/
[6] http://www.cascadelms.org/
[7] http://webwork.maa.org/
[8] http://plone.org/
[9] http://watir.com/
[10] http://seleniumhq.org/

provided by the system is erroneous if they are not able to understand why their submission was judged wrong [26].

- **Compulsory time penalty** after each submission can be used to direct students behavior [1]. Moreover, length of this penalty can grow exponentially after each failed attempt [35].
- **Making each exercise slightly different** is an interesting concept used in QuizPACK by allowing parameterized, automatically assessed random assignments for C programming [8]. Trial-and-error makes no sense when you need to start from scratch to submit again.
- **Programming contests** provide a completely alternative approach where the assignment specification is visible only for a short period of time during which the assignment needs to be completed while competing against time (and others). This approach is adopted to education, for example, in Mooshak [26]. The competition aspect has been proven to be an excellent motivation for the students [41] but also generates a number of problems. How to teach students good scheduling of software development process if they are encouraged to perform as fast as possible at least partly regardless of the quality of the work?
- **Various hybrid approaches** and modifications are also possible. For example, Marmoset [72] supports both unlimited and limited number of submissions. First, there is a public test set to check the basic functionality. These tests can always be executed and repeating submissions are not penalized. Second, there are release tests that can only be asked n-times. Feedback from the release tests is also limited to force students to think before asking tests to be executed.

4.5 Possibility for Manual Assessment

It is often a good idea to combine both manual and automated assessment. Teaching assistants (TAs) can provide extra feedback by manually assessing a submission or they can override the grades, etc. From the tools we surveyed, we were able to identify two levels of manual intervention (no support for manual intervention being the third).

- **To enable the teacher to view the student submissions** is the lightest way to support manual intervention. In this approach, the tool itself does not provide any features for the marking but at least makes it possible to manually assess the same submission. Often the same effect can be achieved by logging into the assessment system and fetching the submissions from the database or filesystem where they are stored. However, supporting this through the AA system makes it possible to separate the roles of TAs from administrators of the AA system.
- **Combining manual and automatic feedback** means TAs feedback and automated assessment can both exist at the same time and support each others. This is supported in Web-Cat [16], for example.

None of the systems clearly described that they would allow TAs to completely override the automatic feedback but we still expect some systems to support this. However, this can easily create confusion among both teachers and learners if the origins of the grade are not transparent.

4.6 Sandboxing

Since the programming assignments are typically graded by running the students' solutions on the server side, securing the server against possibly malicious or just incorrect code is important. A good discussion on the possible attacks against a grading server can be found in [18]. However, as important as this topic is, a large portion of the included articles ignored this. The following approaches to secure execution of students' code were mentioned in the articles:

- **Proper sandboxing.** Relying on existing solutions to securely run programs is a common approach. This can be done by using multiple tools like Systrace (used in EduComponents [3]), linux security module (used in [48]), Java security policy (used in [48]), `ptrace` (used in Moe [46]), and `chroot` (used in CTPracticals [27]).
- **Static analysis.** Security can also be addressed by using custom solutions. For example, Algorithm Benchmark uses regular expressions to try to filter out malicious code [10].
- **Grading on the client.** Some systems deal with sandboxing by doing the grading on the client side in students' own computers. *Mailing It In* [65] uses client's email software to launch tests on client side, whereas E-Commerce virtual laboratory [11] uses Selenium-RC to push tests back to the client that did the submission.

Additional security feature implemented in some systems is to have a different server for running the student programs instead of doing it all on the same machine as the rest of the system. This is done, for example, by EduComponents [3].

In addition to securing AA systems, sandboxing can help when assessing the performance of students' programs. Sandboxes can be configured to limit the available memory or CPU time to ensure assessed solutions are efficient enough (e.g. [27, 79]).

4.7 Distribution and Availability

It is surprising, and quite disappointing, to see how few systems are open-source, or even otherwise (freely) available. In many papers, it is stated that a prototype was developed but we were not able to find the tool. In some cases, a system might be mentioned to be open source but you need to contact the authors to get it.

4.8 Specialty

Quite often the driving force for the development of a completely new tool is a revolutionary idea of something that has not yet been done. Or at least this is the case with tools that get researched and published. Specialities of AA systems identified during the survey included:

- **Automatic assessment of GUIs** has been identified already in the survey of Douce [14] and is still of interest. New systems are still developed [24] and the existing ones are extended to meet the special requirements of GUI exercises [77].

- **SQL** tutoring systems have existed since the late 90's. New systems for this specialty were recognized also in this survey (e.g. [13, 38])
- **Concurrent programming** assignments are often extremely error prone and problems may be hard to detect. Testing concurrency is demanding and specialized tools are developed to help (e.g. [52]).
- **Web-programming** and testing both functionality and security of the websites students implement is getting more attention together with the web-programming getting a stronger position in the curricula. These systems are typically testing a web site (HTML + JavaScript) ignoring how the server side of the site is implemented (e.g. [11, 19, 28, 32, 75]).
- **Letting students do the quality assurance**, either by writing tests for themselves (e.g. [77]) or reviewing code of others (e.g. [61]) is often well grounded to the pedagogical needs.

5. DISCUSSION AND CONCLUSIONS

In this paper, we have surveyed the recent developments of automatic assessment tools for programming assignments. We have done this by systematically collecting relevant articles published in years 2006-2010 to get a sense what has happened in the field since the previous literature reviews on the topic were conducted. The systems included can be roughly divided into two categories: 1) automatic assessment systems for programming competitions and 2) automatic assessment systems for (introductory) programming education.

To answer our first research questions, we have discussed the key features of AA systems in Section 4. From these, we think that the differences in how tests are defined, how resubmissions are handled, and how the security is guaranteed were the most significant.

Based on the data we collected, it is possible to make some recommendations how new AA systems could get more widely adopted. First, we recommend that authors describing new systems should explain more explicitly how the system actually works and provide more examples. For example, instead of stating that the assessment is based on scripts, examples should be provided.

In addition, security of the assessment systems should get more attention. Use of proper sandboxing based on existing security solutions should be encouraged and use of home baked static analysis should be avoided. The latter can leave the system vulnerable, since, for example, the filtering of code using regular expressions is error-prone. Ultimately the security needs to be provided in a way that makes installing the system as easy as possible without compromising security. However, configuring a sandbox can be complicated. Preferably, initial security configuration should not rely on teachers' skills. For example, writing a proper Java security policy is doable (although letting AA system to provide such policies is better) but setting up a secure linux playground with `chroot`, for example, is demanding and teachers might be tempted to make shortcuts. In fact, we believe that lack of sandboxing, or the difficulties in configuring the sandbox, is often one of the obstacles in adopting a system.

The lack of open-source systems might be one of the reasons for the constant development of new tools – that are also likely to remain in-house. We understand people do not want to publish something unfinished but at the same time this slows down new ideas from spreading wider. Thus, we argue that by open-sourcing the existing tools to some popular online version control repository like GitHub[11] or Google Code[12], the tools could be much more willingly adopted by others.

To answer our second research question, we expect new research to emerge from the following fields, from where the first steps were identified in this survey:

- Integrating automatic assessment into LMSs. As another possible path, some of the assessment systems can grow into LMSs if they are modular enough and if they get the momentum behind them. For example, we see that Web-CAT with the various assessment modules already implemented into it is a good candidate to become a CS specific LMS.
- Putting more effort into security of automatic assessment systems. This is also related to the LMS integration because having multiple courses hosted on one platform makes this a more tempting target to hack.
- Automatic assessment of web applications students implement. This can be seen to continue the GUI and SQL testing efforts that have longer traditions. The new aspect we expect to get more importance is security/penetration testing of students' web applications.

There seems to be a steady interest in developing new automatic assessment tools. Sometimes the need to implement yet another system can be challenged and one should ask whether the new feature could be added directly into an existing open source system as in Web-CAT [77], for example. In addition, to increase the adoption of existing systems and thus avoiding the reinvention of the wheel, we strongly suggest automatic assessment system developers to make their systems open source making it easier for others to contribute.

6. FUTURE WORK

Classification presented in Section 4 can be further improved. For example this could be a starting point for a more formal Delphi study [64] with more experts deciding on the categories. Outcome could result in a taxonomy on automatic assessment of programming assignments.

In this survey, we had quite narrow scope. There are many systems closely related to AA we did not cover: systems designed for formative/visual feedback (e.g. [31]), peer review systems (e.g. [71]), and systems to provide feedback on misconceptions and problem solving strategies (e.g. [59]) – to name a few. Identifying the types of systems that can cooperate in an AA setup is essential for understanding how AA systems should be improved from the technical perspective.

Many of the papers we surveyed reported educational experimentations and results of comparing different approaches in automatic assessment. Combining those results with the features of AA systems (presented in this paper) is something we are looking next. This is important for improving AA systems from the pedagogical perspective.

[11] `http://github.com/`
[12] `http://code.google.com`

7. REFERENCES

[1] K. Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005.

[2] A. Alstes and J. Lindqvist. Verkkoke: learning routing and network programming online. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE Conf. on Innovation and technology in computer science education*, pages 91–95, New York, NY, USA, 2007. ACM.

[3] M. Amelung, P. Forbrig, and D. Rösner. Towards generic and flexible web services for e-assessment. In *ITiCSE '08: Proceedings of the 13th annual Conf. on Innovation and technology in computer science education*, pages 219–224, New York, NY, USA, 2008. ACM.

[4] M. Amelung, M. Piotrowski, and D. Rösner. Educomponents: experiences in e-assessment in computer science education. In *ITICSE '06: Proceedings of the 11th annual SIGCSE Conf. on Innovation and technology in computer science education*, pages 88–92, New York, NY, USA, 2006. ACM.

[5] D. J. Barnes and M. Kölling. *Objects First with Java - A Practical Introduction using BlueJ, Thid edition*. Prentice Hall / Pearson Education, 2006.

[6] J. Biggs and C. Tang. *Teaching for Quality Learning at University : What the Student Does (3rd Edition)*. Open University Press, 2007.

[7] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *J. Syst. Softw.*, 80(4):571–583, 2007.

[8] P. Brusilovsky and S. Sosnovsky. Individualized exercises for self-assessment of programming knowledge: An evaluation of quizpack. *J. Educ. Resour. Comput.*, 5(3):6, 2005.

[9] J. Carter, J. English, K. Ala-Mutka, M. Dick, W. Fone, U. Fuller, and J. Sheard. ITICSE working group report: How shall we assess this? *SIGCSE Bulletin*, 35(4):107–123, 2003.

[10] M.-Y. Chen, J.-D. Wei, J.-H. Huang, and D. T. Lee. Design and applications of an algorithm benchmark system in a computational problem solving environment. In *ITICSE '06: Proceedings of the 11th annual SIGCSE Conf. on Innovation and technology in computer science education*, pages 123–127, New York, NY, USA, 2006. ACM.

[11] J. Coffman and A. C. Weaver. Electronic commerce virtual laboratory. In *SIGCSE '10: Proceedings of the 41st ACM technical symposium on Computer science education*, pages 92–96, New York, NY, USA, 2010. ACM.

[12] S. Cooper, W. Dann, and R. Pausch. Alice: a 3-d tool for introductory programming concepts. In *CCSC '00: Proceedings of the fifth annual CCSC northeastern Conf. on The journal of computing in small colleges*, pages 107–116, , USA, 2000. Consortium for Computing Sciences in Colleges.

[13] M. de Raadt, S. Dekeyser, and T. Y. Lee. Do students sqlify? improving learning outcomes with peer review and enhanced computer assisted assessment of querying skills. In *Proceedings of the 6th Baltic Sea Conf. on Computing education research*, pages 101–108, New York, NY, USA, 2006. ACM.

[14] C. Douce, D. Livingstone, and J. Orwell. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.*, 5(3):4, 2005.

[15] S. H. Edwards and M. A. Pérez-Quiñones. Experiences using test-driven development with an automated grader. *J. Comput. Small Coll.*, 22(3):44–50, 2007.

[16] S. H. Edwards and M. A. Pérez-Quiñones. Web-cat: automatically grading programming assignments. In *ITiCSE '08: Proceedings of the 13th annual Conf. on Innovation and technology in computer science education*, pages 328–328, New York, NY, USA, 2008. ACM.

[17] M. Forišek. On the suitability of programming tasks for automated evaluation. *Informatics in education*, 5(1):63–76, 2006.

[18] M. Forišek. Security of programming contest systems. In *Informatics in Secondary Schools, Evolution and Perspectives*, Vilnius, Lithuania, 2006.

[19] X. Fu, B. Peltsverger, K. Qian, L. Tao, and J. Liu. Apogee: automated project grading and instant feedback system for web based computing. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 77–81, New York, NY, USA, 2008. ACM.

[20] G. Gárcia-Mateos and J. L. Fernández-Alemán. A course on algorithms and data structures using on-line judging. *SIGCSE Bull.*, 41(3):45–49, 2009.

[21] O. Gotel and C. Scharff. Adapting an open-source web-based assessment system for the automated assessment of programming problems. In *WBED'07: Proceedings of the sixth Conf. on IASTED International Conf. Web-Based Education*, pages 437–442, Anaheim, CA, USA, 2007. ACTA Press.

[22] O. Gotel, C. Scharff, and A. Wildenberg. Extending and contributing to an open source web-based system for the assessment of programming problems. In *PPPJ '07: Proceedings of the 5th international symposium on Principles and practice of programming in Java*, pages 3–12, New York, NY, USA, 2007. ACM.

[23] O. Gotel, C. Scharff, and A. Wildenberg. Teaching software quality assurance by encouraging student contributions to an open source web-based system for the assessment of programming assignments. *SIGCSE Bull.*, 40(3):214–218, 2008.

[24] G. R. Gray and C. A. Higgins. An introspective approach to marking graphical user interfaces. In *ITICSE '06: Proceedings of the 11th annual SIGCSE Conf. on Innovation and technology in computer science education*, pages 43–47, New York, NY, USA, 2006. ACM.

[25] T. Greening. Computer Science Educational Futures: The Nature of 2020" Foresight. In *Computer Science Education in the 21st Century*, pages 1–6. Springer Verlag, 1999.

[26] P. Guerreiro and K. Georgouli. Combating anonymousness in populous cs1 and cs2 courses. In *ITICSE '06: Proceedings of the 11th annual SIGCSE Conf. on Innovation and technology in computer science education*, pages 8–12, New York, NY, USA, 2006. ACM.

[27] E. Gutiérrez, M. A. Trenas, J. Ramos, F. Corbera, and S. Romero. A new moodle module supporting automatic verification of vhdl-based assignments. *Comput. Educ.*, 54(2):562–577, 2010.

[28] F. Gutierrez. Stingray: a hands-on approach to learning information security. In *SIGITE '06: Proceedings of the 7th Conf. on Information technology education*, pages 53–58, New York, NY, USA, 2006. ACM.

[29] M. T. Helmick. Interface-based programming assignments and automatic grading of java programs. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE Conf. on Innovation and technology in computer science education*, pages 63–67, New York, NY, USA, 2007. ACM.

[30] C. Higgins, T. Hegazy, P. Symeonidis, and A. Tsintsifas. The coursemarker cba system: Improvements over ceilidh. *Education and Information Technologies*, 8(3):287–304, 2003.

[31] C. D. Hundhausen and J. L. Brown. An experimental study of the impact of visual semantic feedback on novice programming. *J. Vis. Lang. Comput.*, 18(6):537–559, 2007.

[32] W.-Y. Hwang, C.-Y. Wang, G.-J. Hwang, Y.-M. Huang, and S. Huang. A web-based programming learning environment to support cognitive development. *Interacting with Computers*, 20(6):524 – 534, 2008.

[33] P. Ihantola. Creating and visualizing test data from programming exercises. *Informatics in education*, 6(1):81–102, 2007.

[34] D. Jackson and M. Usher. Grading student programs using assyst. In *SIGCSE '97: Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*, pages 335–339, New York, NY, USA, 1997. ACM.

[35] T. Janhunen, T. Jussila, M. Järvisalo, and E. Oikarinen. Teaching Smullyan's analytic tableaux in a scalable learning environment. In *Proceedings of the Fourth Finnish / Baltic Sea Conf. on Computer Science Education*, volume TKO-42/04 of *Research Report Series of Laboratory of Information Processing Science, Helsinki University of Technology*, pages 85–94. Otamedia, December 2004.

[36] D. Jimenez-Gonzalez, C. Alvarez, D. Lopez, J.-M. Parcerisa, J. Alonso, C. Perez, R. Tous, P. Barlet, M. Fernandez, and J. Tubella. Work in progress-improving feedback using an automatic assessment tool. In *Proceedings of 38th annual Frontiers in Education Conf.*, pages S3B–9 –T1A–10, oct. 2008.

[37] M. Joy, N. Griffiths, and R. Boyatt. The BOSS online submission and assessment system. In *ACM Journal on Educational Resources in Computing, volume 5, number 3, September 2005. Article 2.* ACM, 2005.

[38] H. Ke, G. Zhang, and H. Yan. Automatic grading system on sql programming. In *Scalable Computing and Communications; Eighth International Conf. on Embedded Computing, 2009. SCALCOM-EMBEDDEDCOM'09. International Conf. on*, pages 537 –540, sept. 2009.

[39] R. Kolstad. Infrastructure for contest task development. *Olympiads in Informatics*, 3:38–59, 2009.

[40] A. Korhonen, L. Malmi, and P. Silvasti. TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. In *Proceedings of the Third Annual Baltic Conf. on Computer Science Education*, pages 48–56, Joensuu, Finland, 2003.

[41] T. Lehtonen. Javala – addictive e-learning of the java programming language. In *Koli Calling 2005 – Fifth Koli Calling Conf. on Computer Science Education*, pages 41–48, 2005.

[42] Y. Liang, Q. Liu, J. Xu, and D. Wang. The recent development of automated programming assessment. In *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conf. on*, pages 1 –5, dec. 2009.

[43] M. Lingling, Q. Xiaojie, Z. Zhihong, Z. Gang, and X. Ying. An assessment tool for assembly language programming. In *Computer Science and Software Engineering, 2008*, volume 5, pages 882 –884, dec. 2008.

[44] L. Malmi, V. Karavirta, A. Korhonen, and J. Nikander. Experiences on automatically assessed algorithm simulation exercises with different resubmission policies. *Journal of Educational Resources in Computing*, 5(3), September 2005.

[45] M. V. Mäntylä and C. Lassenius. Drivers for software refactoring decisions. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pages 297–306, New York, NY, USA, 2006. ACM.

[46] M. Mareš. Perspectives on grading systems. *Olympiads in Informatics*, 1:124–130, 2007.

[47] M. Mareš. Moe – design of a modular grading system. *Olympiads in Informatics*, 3:60–66, 2009.

[48] B. Merry. Using a linux security module for contest security. *Olympiads in Informatics*, 3:67–73, 2009.

[49] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari. Visualizing programs with Jeliot 3. In *Proceedings of the International Working Conf. on Advanced Visual Interfaces*, pages 373 – 376, Gallipoli (Lecce), Italy, May 2004. ACM.

[50] K. A. Naudé, J. H. Greyling, and D. Vogts. Marking student programs using graph similarity. *Comput. Educ.*, 54(2):545–561, 2010.

[51] P. Nordquist. Providing accurate and timely feedback by automatically grading student programming labs. *J. Comput. Small Coll.*, 23(2):16–23, 2007.

[52] R. Oechsle and K. Barzen. Checking automatically the output of concurrent threads. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE Conf. on Innovation and technology in computer science education*, pages 43–47, New York, NY, USA, 2007. ACM.

[53] J. O'Kelly and J. P. Gibson. Robocode & problem-based learning: a non-prescriptive approach to teaching programming. In *ITICSE '06: Proceedings of the 11th annual SIGCSE Conf. on Innovation and technology in computer science education*, pages 217–221, New York, NY, USA, 2006. ACM.

[54] R. E. Pattis, J. Roberts, and M. Stehlik. *Karel the robot (2nd ed.): a gentle introduction to the art of programming.* John Wiley & Sons, Inc., New York, NY, USA, 1994.

[55] A. Pears, S. Seidman, C. Eney, P. Kinnunen, and L. Malmi. Constructing a core literature for computing education research. *SIGCSE Bulletin*, 37(4):152–161, 2005.

[56] A. Radenski. Digital cs1 study pack based on moodle and python. In *ITiCSE '08: Proceedings of the 13th annual Conf. on Innovation and technology in computer science education*, pages 325–325, New York, NY, USA, 2008. ACM.

[57] M. A. Revilla, S. Manzoor, and R. Liu. Competitive learning in informatics: The uva online judge experience. *Olympiads in Informatics*, 2:131–148, 2008.

[58] P. Ribeiro and P. Guerreiro. Increasing the appeal of programming contests with tasks involving graphical user interfaces and computer graphics. *Olympiads in Informatics*, 1:139–164, 2007.

[59] M. M. T. Rodrigo, R. S. Baker, M. C. Jadud, A. C. M. Amarra, T. Dy, M. B. V. Espejo-Lahoz, S. A. L. Lim, S. A. Pascua, J. O. Sugay, and E. S. Tabanao. Affective and behavioral predictors of novice programmer achievement. In *ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE Conf. on Innovation and technology in computer science education*, pages 156–160, New York, NY, USA, 2009. ACM.

[60] G. Rößling and B. Freisleben. ANIMAL: A system for supporting multiple roles in algorithm animation. *Journal of Visual Languages and Computing*, 13(3):341–354, 2002.

[61] G. Rößling and S. Hartte. Webtasks: online programming exercises made easy. *SIGCSE Bull.*, 40(3):363–363, 2008.

[62] G. Rößling, M. Joy, A. Moreno, A. Radenski, L. Malmi, A. Kerren, T. Naps, R. J. Ross, M. Clancy, A. Korhonen, R. Oechsle, and J. A. V. Iturbide. Enhancing learning management systems to better support computer science education. *SIGCSE Bull.*, 40(4):142–166, 2008.

[63] G. Rößling and T. Vellaramkalayil. A visualization-based computer science hypertextbook prototype. *Trans. Comput. Educ.*, 9(2):1–13, 2009.

[64] G. Rowe and G. Wright. The delphi technique as a forecasting tool: issues and analysis. *International Journal of Forecasting*, 15(4):353–375, October 1999.

[65] J. A. Sant. "mailing it in": email-centric automated assessment. In *ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE Conf. on Innovation and technology in computer science education*, pages 308–312, New York, NY, USA, 2009. ACM.

[66] J. P. Sauvé and O. L. Abath Neto. Teaching software development with atdd and easyaccept. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 542–546, New York, NY, USA, 2008. ACM.

[67] J. P. Sauvé, O. L. Abath Neto, and W. Cirne. Easyaccept: a tool to easily create, run and drive development with automated acceptance tests. In *AST '06: Proceedings of the 2006 international workshop on Automation of software test*, pages 111–117, New York, NY, USA, 2006. ACM.

[68] J. Sheard, S. Simon, M. Hamilton, and J. Lönnberg. Analysis of research into the teaching and learning of programming. In *ICER '09: Proceedings of the fifth international workshop on Computing education research workshop*, pages 93–104, New York, NY, USA, 2009. ACM.

[69] M. Sitaraman, J. O. Hallstrom, J. White, S. Drachova-Strang, H. K. Harton, D. Leonard, J. Krone, and R. Pak. Engaging students in specification and reasoning: "hands-on" experimentation and evaluation. In *ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE Conf. on Innovation and technology in computer science education*, pages 50–54, New York, NY, USA, 2009. ACM.

[70] A. Solomon, D. Santamaria, and R. Lister. Automated testing of unix command-line and scripting skills. In *Information Technology Based Higher Education and Training, 2006. ITHET '06. 7th International Conf. on*, pages 120 –125, july 2006.

[71] H. Sondergaard. Learning from and with peers: the different roles of student peer reviewing. In *ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE Conf. on Innovation and technology in computer science education*, pages 31–35, New York, NY, USA, 2009. ACM.

[72] J. Spacco, D. Hovemeyer, W. Pugh, F. Emad, J. K. Hollingsworth, and N. Padua-Perez. Experiences with marmoset: designing and using an advanced submission and testing system for programming courses. In *ITICSE '06: Proceedings of the 11th annual SIGCSE Conf. on Innovation and technology in computer science education*, pages 13–17, New York, NY, USA, 2006. ACM.

[73] J. T. Stasko. TANGO: A framework and system for algorithm animation. *IEEE Computer*, 23(9):27–39, 1990.

[74] H. Suleman. Automatic marking with sakai. In *SAICSIT '08: Proceedings of the 2008 annual research Conf. of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries*, pages 229–236, New York, NY, USA, 2008. ACM.

[75] M. Sztipanovits, K. Qian, and X. Fu. The automated web application testing (awat) system. In *ACM-SE 46: Proceedings of the 46th Annual Southeast Regional Conf.*, pages 88–93, New York, NY, USA, 2008. ACM.

[76] P. G. Thomas, N. Smith, and K. G. Waugh. Computer assisted assessment of diagrams. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE Conf. on Innovation and technology in computer science education*, pages 68–72, New York, NY, USA, 2007. ACM.

[77] M. Thornton, S. H. Edwards, R. P. Tan, and M. A. Pérez-Quiñones. Supporting student-written tests of gui programs. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 537–541, New York, NY, USA, 2008. ACM.

[78] D. W. Valentine. CS educational research: a meta-analysis of SIGCSE technical symposium proceedings. In *SIGCSE '04: Proceedings of the 35th SIGCSE Technical Symposium on*

Computer Science Education, pages 255–259, New York, NY, USA, 2004. ACM Press.

[79] T. Verhoeff. Programming task packages: Peach exchange format. *Olympiads in Informatics*, 2:192–207, 2008.

[80] T. Wang, X. Su, Y. Wang, and P. Ma. Semantic similarity-based grading of student programs. *Inf. Softw. Technol.*, 49(2):99–107, 2007.

APPENDIX

A. LIST OF TOOLS

This appendix briefly describes publicly available (to download or with a demo site where to experiment) tools found in this survey. Many papers did not have explicit statement about the availability of the system, and in many cases we failed to find a site. There were also cases where university's public version control where the system was distributed no longer existed. It should be noted that some of the tools we failed to find could be available by asking the authors. This was not done. References found in this survey and the URL from where more information can be retrieved are mentioned for each system.

AutoGrader [29, 51] is a subproject of Cascade LMS. It has been used to assess Java, but according to authors it can be extended to other languages. Tests are executed through Java reflection similarly to what JUnit does.
`http://www.cascadelms.org/autograder/` (GPL like)

AWAT [75] is an environment for web programming assignments where students only submit an URL of a site they developed. Teacher defines which components should exist on the web-page and tests by using the Watir Ruby library both combined into an Excel sheet. Testing is then performed by using Internet Explorer from the submission server.
Open source, contact authors

CTPracticals [27] is a Moodle module to bring automatically assessed VHDL exercises into Moodle. External test script and sandboxing are both configurable through the Moodle UI. The framework can also be extended to other programming languages. For example, there are Matlab exercises on the demo site.
`http://guac.ac.uma.es/demo` (login credentials in [27])

EasyAccept [66, 67] framework provides a natural-language-like scripting language to write tests for Java programs. Requirements are presented in a form of acceptance tests.
`http://easyaccept.sourceforge.net/` (GPL)

EduComponents [3, 4] is a set of components to the Plone CMS for creation, management and assessment of programming assignments. It has different backends for different programming languages which allow (depending on the language) unit testing, comparison to a model answer or more formally defined testing.
`http://plone.org/products/ecautoassessmentbox/` (GPL)

Linuxgym [70] supports exercises and examinations of unix scripting skills. An extensive exercise definition language is also included.
`http://linuxgym.com/` (GPL)

Moe [46, 47] (originally MO-eval) is a modular environment for programming contests with sandbox, queue manager, and submitter for managing submissions, and different graders (that can be combined). The aim is make various modules interchangeable.
`http://mj.ucw.cz/moe/` (GPL2)

Mooshak [20, 26, 58] has its origins in programming contests, although it has also been used in teaching. One of the specialties of Mooshak is that results of the assessment can be publicly shown to other students.
`http://code.google.com/p/mooshak/` (Artistic License/GPL)

Peach[3] [79] is a highly configurable system for programming education and contest hosting.
`http://peach3.nl/` (Artistic License v2)

ProtoAPOGEE [19] is a prototype of a proposed system to grade web sites like AWAT. APOGEE relies on Watir test library and it can also take series of screenshots from the web site being assessed. Screenshots can then be used as feedback to explain why a test failed.
`http://vlab.gsw.edu/Projects/APOGEE/` (sources and video for academic research or evaluation)

Resolver [69] combines formal verification and traditional programming assignments. There are exercises where students need to demonstrate their understanding of formal specifications by writing tests and exercises where students write programs verified against formally expressed contracts.
`http://www.cs.clemson.edu/~resolve/` (GPL3)

RoboCode [53] supports Java and .NET assignments where students' programs compete with each other. Grades can be based on the results of the competition.
`http://robocode.sourceforge.net/` (Eclipse Public License)

USACO's [39] competition hosting environment has been developed by the USA Computing Olympiad. The system also offers web based problem development tools to aid in creating competition problems.
`http://train.usaco.org/usacogate` (demo)

UVA Online Judge [57] is mainly intended as a programming contest training site. Users can practice on the large number of existing problems and submit their answers in multiple languages. It is also used for hosting online programming competitions.
`http://uva.onlinejudge.org` (demo)

VERKKOKE [2] is an online teaching environment for socket programming/routing. It generates individual programming assignments which the student completes and submits. One of the specialties is the SCORM integration with LMS systems (e.g. Moodle and Optima).
`http://www.tml.tkk.fi/Research/VERKKOKE/` (MIT)

Web-CAT [15, 77] is a system where students are required not only to submit source code, but also unit test their own code. Part of the grade is based on the test coverage achieved by students' own tests. Web-CAT has a plugin architecture for different graders, static analysis, support for other languages, etc.
`http://web-cat.cs.vt.edu/WCWiki` (Affero GPL)

WeBWorK-JAG (Java auto-grader) [21, 22, 23] is an extension module to the WeBWorK exercise delivery platform. The module allows checking Java programs with JUnit.
`http://csis.pace.edu/~scharff/webwork`