A TECHNICAL PERSPECTIVE ON ASAP – AUTOMATED SYSTEM FOR ASSESSMENT OF PROGRAMMING

Christopher Douce, David Livingstone, James Orwell, Steve Grindle and Justin Cobb

A Technical Perspective on ASAP – Automated System for Assessment of Programming

Christopher Douce, David Livingstone, James Orwell,
Steve Grindle and Justin Cobb

Kingston University
Faculty of Technology
Penrhyn Road
Kingston-upon-Thames
Surrey KT1 2EE

Abstract

To learn computer programming, students are invariably asked to complete some form of assignment, which is often assessed by the instructors. This assessment can be time consuming, and an automatic system of assessment can reduce this burden and allow additional functionality. One practical issue is how to integrate them with the other components of the learning management system used by any given institution. ASAP is an automated programming assessment tool which conforms to the JISC e-learning framework, designed for to make such components interoperable and reusable. This paper reviews the previous work on automatic programming assessment, and then presents a technical review of ASAP, discussing its architecture and standards. The paper then discusses some of the challenges that have been faced in developing tests and running foreign code submitted to a web service. Possible extensions to the system are presented, and the current work is described.

Introduction

Teachers of computer programming and software design frequently need to deliver practical assessments to their students. This allows students to demonstrate their skills and test their own abilities, and allows instructors the chance to evaluate the performance and knowledge of the students.

Often, these exercises are administered by hand. The assessment is written, then delivered as coursework, in a workshop session, or as a traditional closed book exam. Answers are submitted, which have to be tested and marked individually. These three assessment stages can be described as development, delivery and grading. To assist instructors in these three tasks Kingston University have developed ASAP – Automated System for Assessment of Programming.

The first section of this paper reviews prior systems used in automated testing of programming, and what can be learnt from previous technical approaches.

Next we discuss the architecture of ASAP, how it relates to the JISC e-learning framework and other standards, and the decisions taken regarding the implementation of the system. We then cover the details of the development and deployment of test classes by instructors, and some of the technical hazards of testing this way. We conclude by looking at future developments, including a project currently underway which extends ASAP.

This paper explores ASAP from a technical standpoint, A discussion of more pedagogical implications, including student and staff evaluations, can be found in Douce et al. (in press).

Previous Work in Automated Programming Assessment Systems

The earliest automated assignment testing system may have been developed by Hollingsworth (1960). Rather than using compilers and text editors students submitted programs written in assembly language using punched cards. When a grader program was run against a student program, two values could be returned, either 'wrong answer' or 'program complete'. At the time of writing, the advantages of an automatic system were considered not only in terms of tutor resources, but also in efficient use of computing time which allowed student numbers to learn programming.

As programming systems evolved, so did assessment systems. Forsythe and Wirth (1965), along with Naur (1964) present a grader system based upon Algol. The grading programs are said to supply test data, keep track of running time and keep a 'grade book'. In the Forsythe and Worth system every assignment requires a corresponding test program. The operation of the test program and the subject program were then compared. For every test program, a corresponding grader program is created. Low-level equivalents of stream redirect instructions were written to allow a grader to supply values to the program under test. The principle of individual test programs for each submission is something that can be seen to continue throughout the development of automated assessment systems, up to and including ASAP.

Hext and Winings (1969) propose interesting new ideas. Tests are performed by comparing a stored test data value against values obtained from the submitted programming assignment. Following assessment a clearly laid out listing of results is produced.

Developments in technology naturally introduce changes to the testing approaches. Rather than allowing students to view the results of the tests directly, Isaacson and Scott (1989) present a script based system that iteratively processes a number of submissions executing each one against test data whilst creating a report.

Reek (1989) presents the TRY system which allows a student to execute test programs from a command line. Following a test, students are presented with the results and details of each execution attempt are recorded. Reek makes an interesting point which we had to take into account; running alien code in a real environment is dangerous.

Kassandra (von Matt, 1994) facilitates the automatic testing of programs written using the mathematical languages Matlab and Maple, as well as Oberon, the successor to Modula-2. Correctness is again tested by comparing output data with stored test data. The development of the test software is deemed to be something that the tutor should perform. Kassandra provides two separate executable elements – a student component and an 'assistant' component.

The ASSYST system developed by Jackson and Usher (1997) includes a sophisticated assessment scheme that analyses C programming assessments across a number of dimensions, namely whether they are correct (according to some predefined test data), whether they are efficient in terms of CPU time and have sensible ratings of complexity and style. One of the greatest contributions this project makes is the understanding that the assessment system can also become a 'grading support system' comprised of mechanisms to handle submissions, creating and generating reports and allows weightings to be assigned to particular tests.

Many ideas found within these early systems can also be seen within BOSS (apparently an abbreviation for Bob's Own Submission System) from Warwick University (Joy & Luck, 1998). The first version of BOSS consisted of a set of command line programs. Using one program the student could test their assignments written in the C language to determine whether or not they were deemed to be correct. When satisfied they could then use a different utility to submit the program to a secure location. A tutor would then use another program to view and re-test the submissions.

Like many computer science departments, Warwick has started to teach programming using the Java language. To automatically test Java software a redesign to BOSS was required. The resulting system comprises of three main elements – an assignment submission and testing program in the form of a Java application, a tutor grading and assignment management application and a web-based application which combines the two components together. Results from executing tests and details about student records are stored within a relational database.

One significant innovation that BOSS introduces lies with its adoption of the JUnit testing paradigm (Beck, 2003). It was understood there were problems associated with the simple processing of raw input and output of data streams. By applying a method-oriented testing approach, design of assignments could be checked. Secondly, the type of unit testing that JUnit utilises is becoming increasingly adopted within industry and in some cases test-driven development is also being introduced into the classroom. There is another aspect to the BOSS system that is particularly interesting. Whilst the assessment of assignments may be automatic, the allocation of grades continues to lie firmly in the hands of the educator.

Up to this point, all the programs considered have been command line or console based. JEWL (English, 2004) wishes to address assignment submissions of greater substance by developing a system which analyses GUI programs. One of the reasons for this being to improve student motivation since GUI applications are often viewed like real applications, as opposed to toys. The JEWL system is in fact a GUI tool kit where the GUI can be replaced by a test harness which can then interpret instructions that

that program under test executes. Interactions are carried out by a simple ‘message-loop’ system.

A final issue considered is one concerning implementation and deployment. Almost all the systems described within this section are self-contained, each having been constructed in isolation from other systems such as institutional wide e-learning systems or enterprise level administration and admission tools. One exception is BOSS, which has been explicitly designed to interface with a proprietary admissions system. Whilst BOSS can be made to interface with other systems, it is not immediately possible without rework to elements of the available source code.

ASAP Design, Architecture and Implementation

The ASAP project is a system which automatically assesses students programming submissions though a user agent such as a Virtual Learning Environment (VLE), an institution’s online educational environment, providing access to learning materials and tools. ASAP is funded by the JISC e-learning tools strand, an initiative to provide the UK and Higher Education establishments with a series of freely available resources and tools. ASAP fits into an abstract framework which is known as the e-learning framework (or ELF).

The framework is intended to guide the construction and development of reusable software components which can be combined together to meet the requirements of a particular education institution. The framework comprises of a number of *bricks*. Links between bricks are established through the adoption of web-services. The intention is not to replace existing e-learning systems in their entirely but to add a series of services which are intended to associate to a set of perceived needs (Wilson et. al., 2004).

The ASAP project has been mapped to a number of ELF bricks. The most significant bricks being *VLE*, *Web Portal*, *Assessment*, *Grading* and *Authentication*. The definitions for these components are still emerging and will be subject to change as an implementation is associated with each brick. Following this idea the ASAP system adopts a modular format that utilizes web-services. The main components of the ASAP architecture are illustrated in Figure 1.

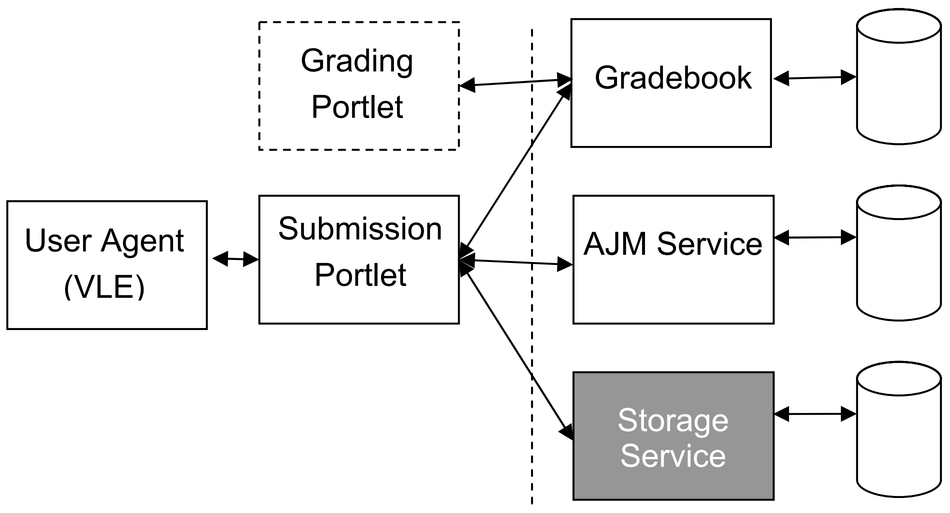

Figure 1 : ASAP Architecture

The ASAP project fits into the e-learning framework as an assessment tool. As other tools are developed, for example gradebooks, authentication systems and item banks, they can be accessed together through any user agent. Here we have discussed Blackboard and uPortal, but alternative implementations could be created, for example using the proprietary VLE Web CT, or the open source VLE Moodle.

The user agent block represents a VLE. A student uses his or her e-learning account to submit a programming assignment. This submission mechanism is represented by the *submission portlet* block.

The ASAP system was initially developed to use the Blackboard VLE. The submission component was developed using the proprietary interface provided by Blackboard. This was used to create the *building block*, which is the name given to extensions to the Blackboard system. The concept of a 'pluggable component' was also applicable when the ASAP project targeted an open source portal named uPortal which uses a specification known as JSR-168.

When a student has submitted an assignment, their program is graded and assessed by a separate software component. The grading engine has been implemented as a web-service. The *submission portlet* sends the source file to the AJM Service, an abbreviation of Automatic Java Marker (see Figure 2). The AJM service receives the program, selects an appropriate test routine and then runs the selected program against the submitted program. The result of each test is collated and an XML document describing the success of the test is then constructed. This document contains comments about the program, a description of tests applied and a final grade. It is up to the submission portlet to decide how this information is used when giving feedback to the student and potentially the score may be rendered into alphabetic grades depending upon individual requirements. Secondly, a system-wide grade book may also be updated giving the tutor a way to view the test results. The logic of grade recording is held within the *submission portlet* and the current Blackboard implementation records only the highest score that a student receives for a particular assignment.

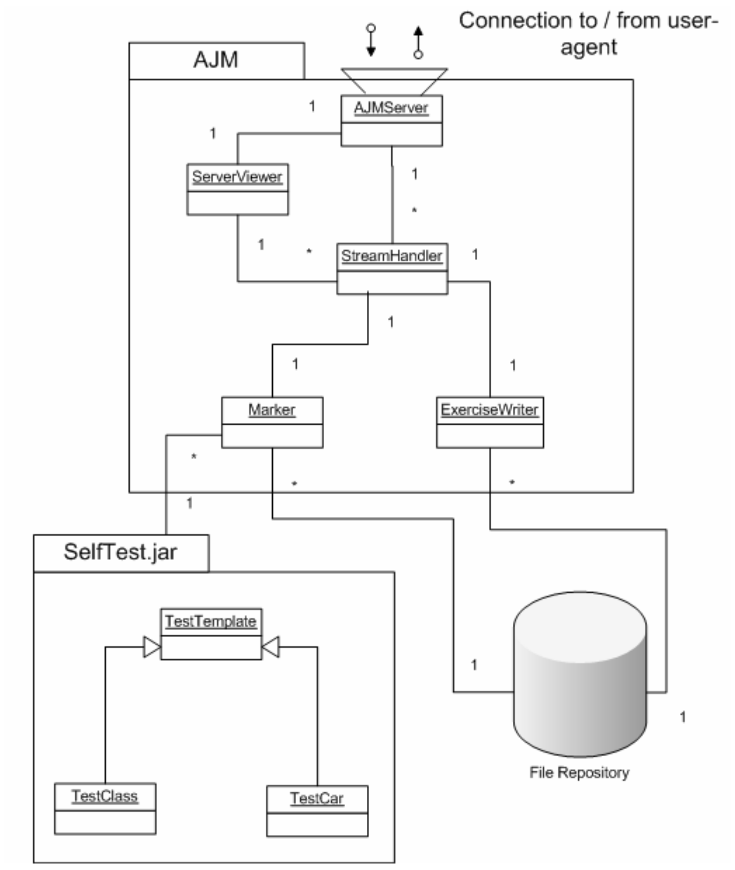

Figure 2 : Automated Java Marker

A number of interoperability specifications are directly relevant to the ASAP project, specifically the content packaging (Smythe, 2002) and the question and test (QTI) specifications (Smythe, 2005).

The content packaging specification describes how materials can be moved from one system to another. The QTI specifications describe computer administered assessments. It is interesting to note that the potential application of QTI was considered by English (2002). The latest version of the QTI specification introduces an additional element that allows the use of an extensible *responseProcessing* mechanism which can be potentially used in association with the new file upload capability.

As assessment initiatives and technologies evolve so will the standards. Whilst it is not yet practical to implement a standards based programming assessment system, the adoption of standards has been considered and will continue to be considered as they change and develop.

ASAP is currently being used on a first year programming module at Kingston University, and has been made available to approximately 140 students.

Testing using ASAP

One of the purposes of ASAP is to assist in the development of tests. When an assessment is written, a test class must also be created for that particular assessment. The three purposes of the test class are to evaluate the submission against some objective criteria, provide feedback on the performance against these criteria, and provide a single mark which can be used to evaluate the overall performance of the submission.

As a test class has to be created for each assessment, this process needs to be made as easy as possible, to reduce the amount of training needed for authors new to the system to be able to develop their own tests.

In early prototypes of Automated Assessment Systems, the test class could be of any form, provided it carried out tests on the submission and produced some output. There was no standard format for tests.

In order to standardise the tests, a test template was created. This takes the form of an abstract superclass that all test classes must inherit from, thus enforcing some behavior on the test classes. The test template also standardizes the way that submissions are compiled, and the format of the final mark. This standardisation allows for greater reuse of tests between years, and between different user agents who can be sure that they will receive output in the same format each time.

The authors have much greater flexibility on two areas: the individual objectives that are tested for, and the feedback that comes with success or failure at each objective. In this example, the author is testing the function of a class to estimate the value of pi. The objective can allow a range of accuracy, rather than demanding a precise answer. The feedback could either be very simple (objective passed, or objective failed) or more detailed (detail the incorrect result produced in the case of failure, and provide hints to the right answer).

Work is underway on the next stage, a test creation wizard, which will allow a test class to be generated from a 'model solution' though a user interface, without the author ever needing to directly write the code for the test class.

There of course some issues and hazards that need to be dealt with when students are able to upload source files for testing. These fall into two categories, 'upload issues' and 'logic issues'. Both concern the safety and security of the system, i.e. accidental and intentional mis-use.

The 'file issues' include scenarios such as the user trying to upload a file that not a valid java source code file, and also files that are too big to be valid. The files are checked to see if the file extension is 'java' by a JavaScript within the web browser. The next test is during upload to the AJM server the size of the file is tested, if it greater than 64KiB the file is rejected since java source code files 'should not' be that big. At the same time the value of the characters in the file are checked, if any of them not plain text characters the file is also rejected since source files should not contain such characters. The final test that is performed is a check that all the files needed for a given exercise are submitted at the same time. If all tests are passed, the process is allowed to continue on to the next stage of testing: the compilation and execution of the student source code by the test harness.

Two logical issues handled in the execution stage are: the non-terminating process, and the output of HTML tags from the process; both are described below. The infinite loop problem is solved by allowing the students' code a fixed execution time, e.g. ten seconds. If it has not finished in this interval then the test is terminated and an error reported back to the student. The output of HTML tags from the student code is filtered by the system. The 'angle brackets' are replaced by HTML 'escape codes' so they appear correctly on the final web page, but do not get processed as tags. The access to the host file system is to be limited by the end user setting up the necessary 'sand-box' security around the web service container.

Further Development and Conclusions

ASAP is continuing to be used at Kingston University and additional members of staff are using the tool to develop tests. Trials have also taken place at City University and De Montfort University.

The ASAP project is currently being expanded on by two further projects. The Jelfad project (JISC e-learning assessment demonstrator) incorporates the system into a larger demonstration project within the ELF, including other JISC e-learning tools and standards such as Content Packaging and Simple Sequencing.

The PAINTS project (Programming Assessment Integrated Training System) directly builds on the work of ASAP by developing a production level system that can stand entirely alone, incorporating content management, assessment management, and an appropriate user agent. Additional tools are being developed at a prototype level. PAINTS will also address the issue of testing higher level program design and design patterns.

Many universities and colleges have adopted some form of Virtual Learning Environment (VLE) to facilitate communication between educators amd students. This represents an opportunity for the introduction of standards, to allow collaboration between institutions, and introduction of novel teaching and learning approaches. VLE systems may be commercial products, open source equivalents or in-house systems. ASAP goes some way to realising a solution that could potentially work in all three cases by using the e-learning reference model as a basis.

Copying other students' work is a perennial problem to educators; particularly for computer science coursework. A programming plagiarism detection system, entitled JPlag, has been developed at the University of Karlsruhe. This system accepts a batch of submissions and generates a sophisticated report detailing similarities and differences between each submission. As a part of an on-going project, a web-service front end to the system is under construction, allowing a VLE, the ASAP Java marking engine and the JPlag detection system to provide the beginnings of an integrated system.

This paper has presented a system for automated assessment of programming assignments, and explained its architecture, how it fits into the e-learning framework, and related standards. We have described two of the most important technical areas in the project development. Firstly, we discussed the problem of creating test classes, and described how it has been possible to standardise this and make it easier for the author. Secondly we dealt with some of the potential problems of allowing submissions of unknown code to be compiled and run.

There are interesting parallels that can be drawn between the ASAP project and other e-learning assessment initiatives. Evidence for these can be seen with the development of QTI and content packaging specifications, and the decision to allow ASAP to be accessed through any VLE.

It is expected that automated assessment systems similar to the ASAP system will increasingly adopt a greater number of e-learning interoperability standards are they become more flexible and cater for a greater number of different user scenarios. Automated testing is particularly relevant for 'test-first' software development strategies. As the computing science and software engineering curriculum embraces new developments in academia and industry, testing regimes and approaches will also change.

Acknowledgements

The ASAP project has been funded by the JISC e-learning tools development strand.

References

Beck, K. (2003). *Test driven development: By example* Addison-Wesley, Boston, MA.

English, J. (2002) Experience with a computer-assisted formal programming examination. ACM SIGCSE Bulletin, *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, 51-54.

Douce, C., Livingstone, D., Orwell, J., Grindle, S and Cobb, J (in press) Automatic Assessment of Programming Assignments, *Proceedings of ALT-C 2005*

Forsythe, G. E., Wirth, N. (1965) Automatic grading programs. *Communications of the ACM*, **8**, 5, 275-529.

Fincher, S. and Petre, M. (2004) *Computer science education research*. Taylor & Francis.

Hext, J. B. and Winings, J. W. (1969) An automatic grading scheme for simple programming exercises. *Communications of the ACM*, **12**,5, 272-275.

Higgins, C., Hegazy, T., Symeonidis, P., and Tsintsifas, A. (2003) The CourseMaster CBA System: Improvements over Ceilidh. *Journal of Education and Information Technologies*, **8**, 3, 287-304.

Hollingsworth, J. (1960) Automatic graders for programming classes, *Communications of the ACM*, **3**, 10, 528-529.

Horstmann, C. (2002) Big Java. John Wiley & Sons Inc.

Isaacson, P. C. and Scott, T. A. (1989) Automating the execution of student programs. *SIGCSE Bulletin*, **21**, 2,15-22.

Jackson, D. and Usher, M. (1997) Grading Student Programs using ASSYST. Technical Symposium on Computer Science Education, *Annual SIGCSE Conference on Innovation and Technology in Computer Science Education,* 335-339.

Joy, M. and Luck, M. (1998) Effective electronic marking for on-line assessment**.** *Proceedings of the 6th annual conference on the teaching of computing*, Dublin City Univ., Ireland, 134-138.

Naur, P. (1964) Automatic grading of students' ALGOL programming. *BIT 4*, 177-188.

Reek, K. A. (1989) The TRY system – or – how to avoid testing student programs. *SIGCSE Bulletin*, 112-116.

Rosbottom, J. (1997) Computer managed, open question, open book assessment. *SIGCSE Bulletin*, 100–102.

Smythe, C., et. al. (2005) IMS Question & Test Interoperability, Version 2.0, *IMS Global Learning Consortium*.

Smythe, C., et. al. (2004a) IMS Content Packaging Best Practice and Implementation Guide v1.1.4, IMS Global Learning Consortium, Inc.

Wilson, S., Blinco, K. and Rehak, D. (2004) Service-oriented frameworks: Modelling the infrastructure for the next generation of e-learning systems. *JISC-CETIS*.

Woit, D. and Mason, D. (2003) Effectiveness of on-line assessment. *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, 137-141.

von Matt, U. (1994) Kassandra: The automatic grading system. *Technical Report UMIACS-TR-94-59*, Institute for Advanced Computer Studies, Department of Computer Science, University of Maryland, USA.