Submit! A Web-Based System for Automatic Program Critiquing

Yusuf Pisan, Debbie Richards, Anthony Sloane, Helena Koncek and Simon Mitchell

Department of Computing
Division of Information and Communication Sciences
Macquarie University
North Ryde, NSW 2109

{ypisan, richards, asloane}@ics.mq.edu.au

Abstract

This paper presents the Submit! project which aims to enhance teaching and learning in computing by developing automated web-based tools that assist in providing critical feedback to students about the computer programs they write. By developing sophisticated computer-based tools that will improve our monitoring of student progress and maintenance of consistent standards we aim to provide structured assessment with a level of detail and consistency that would be difficult or impossible to provide manually. By allowing students to use the critiquing tools before final submission of an assignment we offer formative assessment that supports self-directed learning. Submit! has been integrated into many computing units at Macquarie University. Usability evaluations show that Submit! is generally effective while needing improvement in certain areas. Two studies involved a review of the system in terms of its intuitiveness, look, feel and flow. A preliminary study of the impact of Submit! on student results shows that students who make use of the system to get feedback on assignment submissions do better than those who don't.

Keywords: learning computer programming, integrated learning environments, code reviews

1 Introduction

This paper presents a project to enhance teaching and learning in computing by developing automated web-based tools that assist in providing critical feedback to students about the computer programs they write.

By developing sophisticated computer-based tools that will improve our monitoring of student progress and maintenance of consistent standards we aim to provide structured assessment with a level of detail and consistency that would be difficult or impossible to provide manually. By allowing students to use the critiquing tools before final submission of an assignment we offer formative assessment that supports self-directed learning.

The tools, collectively known as the *Submit!* system, are becoming an important part of the on-line support for our units, providing a means of giving students the same quality of assessment regardless of the delivery mode or class size.

In Sections 2 and 3 we provide the background and rationale of the project. Section 4 gives an overview of the project development and its current status. Section 5 discusses results from evaluations that we have performed to date: usability studies from both student and academic perspectives, and an examination of the effects of the system on student performance on a first year assignment. Our conclusions and future work are given in Section 6.

2 Background and Approach

Most computing units use programming tasks to reinforce lecture and tutorial material with practical experience. *Program critiquing* refers to the process by which students obtain critical feedback about their programs. Critiquing can be used as a component of formative or summative assessment to enhance self-directed learning. In fact, providing timely and high-quality feedback has been identified as one of the critical factors in learning (Ramsden, 1988; Gibbs, 1994; Hattie, Jaeger and Bond, 1999).

The current and traditional method of providing critiquing is to do it manually: tutors or lecturers read submissions, optionally run programs on test data, and make comments. In large computing units, such as those in first year where there can be close to one thousand students enrolled in a unit, the amount of time required for critiquing makes it prohibitive to give feedback to students on more than two or three assignments per semester. As a result, students have limited opportunities to learn from their mistakes, and it is difficult for lecturers to monitor student progress.

The alternative to manual critiquing is to automate the process. Current commercial systems are restricted to true/false or multiple-choice questions and are of limited use for deeper learning. A number of intelligent learning environments have been developed as research prototypes. Some well-known examples are SOPHIE (Brown et al., 1982), LISP Tutor (Anderson and Reiser, 1985), Geometry Tutor (Anderson et al., 1986), Sherlock (Lajoie and Lesgold, 1989), WebToTest (Arnow and Barshay, 1999), and PILOT (Bridgeman et al., 2000). Much can be learned from these systems in terms of underlying techniques; however, they are limited to specific domains and, most importantly, do not address the program critiquing problem.

Although a variety of testing and critiquing programs had been developed within the department, these

programs have lacked a uniform interface, been difficult to use by people other than the original author, and have often duplicated previous efforts.

The goals of the Submit! project are:

- to develop a suite of tools that specifically support program critiquing;
- to provide a uniform interface to these tools; and
- to ensure that the tools are designed with usability in mind.

The process of critiquing programs and generating high-quality feedback is a complex task. Despite this complexity, program critiquing is feasible because each programming language has a well-defined syntax and semantics that makes the task of automation easier than it would be for free-form natural language submissions such as essays or reports.

3 Educational Rationale and Conduits to Student Learning

We expect automated program critiquing to deliver two major benefits to students: improved feedback and more useful access to lecturers.

Feedback will be better because:

- It will be specific to the individual's submission rather than a general list of possible errors or solutions that the student is often unable to apply to his or her own case.
- There will be more feedback as comments can be added every time the need is identified, in contrast to the current circumstance where the quality of the commenting varies as a result of marker fatigue and time pressure. The feedback will be more consistent and objective as the computer will always provide the same detailed feedback regardless of whether it is the first or the last student who has made the mistake.
- The feedback will be more timely because the computer will be able to process solutions quicker than a human.
- Formative feedback will be possible because in some cases students will be able to run the critiquer themselves before final submission to allow them to correct their own mistakes. There is more incentive for students to spend the time understanding what they did wrong when it can have a direct impact on their marks. Feedback offered after marking is often not incorporated or even read, as evidenced by the number of assignments that never get picked up. Also, students often complain that they did not understand what was expected of them. Being able to test out a solution before submission should reduce this problem. Advanced students will be able to proceed to the next set of exercises and get feedback on their progress instead of being forced to wait while the topic is covered in lectures.

Access to lecturers will be more useful because:

- The more mundane questions such as "What did I do wrong?" will be answered by the critiquer. This means that lecturer-student interaction can concentrate on the concepts underlying the solution.
- Freeing lecturers from some of the marking task will result in an increase in the time available for consultation with students.
- Students and lecturers should find interaction more satisfying as the nature of the discussions should be more stimulating and less shallow.

The project will also provide substantial benefits to our department:

- The department has an ongoing plan for quality improvement throughout its curriculum, based on standardisation and documentation of procedures across all units. By assisting in the enforcement of consistency in assessment, the project addresses an issue that significantly affects the quality of teaching and learning in Computing at Macquarie.
- The project will lead to a reduction in our reliance on casual staff and the consequent costs. Some casual staff assistance will always be necessary to assist in assessment, but as students take a more self-directed learning approach in computing units, these staff will be able to focus on more productive assistance rather than routine assignment marking.
- The generality of the system we are developing means that a broad range of computing units will benefit as a result. We are not focussing on one particular programming language, since the Department of Computing uses a variety of programming languages, and the range used changes over time. Our general approach and system will thus support a wide range of units as well as allow the department to keep up with the latest trends in computer programming languages.

4 Project Development

4.1 Overview

The project is:

- **building** an automatic critiquing system for computer programs;
- **deploying** the system in selected Computing units at Macquarie; and
- **evaluating** the effectiveness of the system.

The basic functionality of the system includes acceptance of submissions from students in the form of files containing program code, analysis of them according to specific criteria, and return of an analysis report to the student. A web-based interface has been created so that the system can be accessed from anywhere.

Our approach to the project has been an incremental one, whereby at any point we only provide as much automatic critiquing as we can carry out reliably. For example, our first step was a simple critiquer that runs the program on a standard dataset and alerts the student

if the program does not produce expected output. This does not release the students from choosing an appropriate dataset to test their programs. The lecturer can set the feedback to students to include the test dataset or just provide hints to the fact that there might be additional inputs to their program that they do not currently handle. As the work progresses, we are incorporating increasingly sophisticated critiques. This methodology, built on top of a consistent framework and interface, means that we are deriving benefits from the project at an early stage.

4.2 Design Considerations

A number of issues have been considered in the design of the system:

- Development of mechanisms to handle the diverse aspects of the critiquing problem, including which tests to run, which analyses to apply to the program, what form the feedback should take, and, if applicable, how to compute marks.
- Identification of the different forms of critiquing possible and their relative usefulness to lecturers and students. Examples include help with syntax errors that are commonly made by beginning students, and special tips that take their experience with a language in earlier units into account when learning a new language. It is also useful to incorporate suggestions that do not address errors *per se* but guide students towards better programming styles. Our goal was to develop tools that instructors can modify to suit their needs rather than agree on a *preferred programming style*. Tips include: comments on indentation, length of procedures, organisation of procedures, determining whether a function is recursive or iterative, and identifying dead code that is not used but left over from earlier stages in the development cycle.

We recognise that the more mundane aspects of automation such as setting due dates, accepting late submissions, allowing only files with a specific name has to be controlled and easily accessible to lecturers. Our incremental approach has ensured that staff have been able to evaluate the program critiquer and provide feedback during the development phase.

4.3 Project Phases

The project consists of four phases: requirements specification (Phase 1), development and evaluation of an initial prototype (Phase 2), extended system development (Phase 3), and final evaluation (Phase 4). The activities under these phases involve some temporal overlap. The content of the phases is described in detail below. We are currently in Phases 3 and 4.

4.3.1 Phase 1: Requirements Specification

Many staff members have used ad hoc computer-based testing in their units, and these systems represent a valuable source of ideas and prototypes. More importantly, staff using these systems have knowledge of likely problems and difficulties. Informal student feedback received about the various ad hoc systems used in the past has been overwhelmingly positive.

In this first phase of the project, we determined the detailed requirements for the system by interviewing staff and students. This phase had three main strands of activity:

- First, we held a number of open sessions where staff members and students could provide input in a shared group environment.
- Second, we conducted one-on-one interviews with the convenors of every unit taught in the department.
- Third, and most importantly, we ran the prototype system developed in Phase 2 in parallel with the existing assessment system in two core units throughout the semester, to determine how best the program critiquer could be integrated into these units. We chose a first year unit with an enrolment of over 200 students and which was taught in both semesters, so that it could be used for the prototype testing and evaluation in the second half-year. The other unit was a smaller second year unit which we chose to ensure the needs of units of different sizes were adequately considered.

4.3.2 Phase 2: Initial Prototype + Evaluation

Using the requirements derived in the early parts of Phase 1, we developed an initial version of the system, focussing on those aspects of program critiquing we identified as being of most value. The basic functionality of this version of the system was relatively simple as we were concerned to produce a usable tool that would allow us to test the overall structure and development methodology. The system was initially targeted for deployment evaluation in two units, again one large and one medium-sized.

- Phase 2A of the project encompassed the basic system construction.
- Phase 2B was the deployment phase, where we monitored the system in use and fine-tuned as appropriate.
- Phase 2C constituted the first evaluation phase of the project, wherein we assessed the success or otherwise of the initial prototype. Feedback was sought from both students and staff. We include a description of the usability evaluation and some of our results in Sections 5.1 and 5.2.

Once students and lecturers had gained some experience with the system, we allowed students to use the system before submitting a final solution to determine the benefits of using it for detecting and correcting errors before submission. We have conducted a preliminary evaluation to assess the value of prior self-assessment of assignments in an important first year unit. Our study compared the quality of student submissions this semester with submissions by students in the previous semester who did not have this

facility. This study and our conclusions are described in Section 5.3.

We will also provide self-assessment for the other assignments in this unit. At the end of the semester we will review examination results to see if there is a marked difference in overall performance. Of course, we will need to take into consideration other factors that may have affected student performance. The initial system development used PHP to generate dynamic HTML based web pages which has worked well for this project.

4.3.3 Phase 3:Extended System Development

In this phase, the initial prototype is being extended in a number of ways.

- We are feeding the experiences gained from the initial evaluation back into the system to fix any problems discovered.
- We are extending the range of critiquing available, ensuring that the needs of a wider range of units are adequately catered for.

Currently the basic system architecture supports automated testing facilities and some units are taking advantage of these facilities. While an extensive library of tests that instructors can use is under development, the library is meant as a series of examples that would need to be modified based on the needs of lecturers.

There will be a major jump in functionality from the initial prototype to the extended system, mostly in the sophistication of critiquing that is supported, and so a significant development period is required for this phase. However, while this development is ongoing, incrementally refined versions of the system are being released in a progressive fashion so that the software can continue to be used throughout the teaching term.

4.3.4 Phase 4: Final Evaluation

The extended system is being evaluated in an ongoing fashion in a variety of units. When its functionality has largely been frozen, we will conduct a full-scale evaluation of the system in at least four mainstream Computing units at a variety of levels.

By this phase of the project we expect that the system will be integrated into most of the units within the department. Already its use is widespread and we are conducting workshops at the beginning of each semester to update staff on its capabilities and help them use it in their units. These workshops are also providing a valuable feedback channel for future enhancements.

5 Monitoring and Evaluation

The project plan mapped out in Section 4 indicates how evaluation fits into the project structure. Evaluation is to be conducted in all phases of the project, and is the primary focus of Phase 4. Our evaluation of the tools developed is based on four criteria:

1. the usability of the tools;
2. the attitudes of both staff and students towards the tools;
3. the impact of the tools on students' scores in programming tasks; and
4. the impact of the tools on students' deeper knowledge of course content.

To date we have conducted evaluations to assess the first three criteria: two usability studies, one from the student perspective and one from the staff perspective and an initial evaluation of the impact of the tool on student scores. The design and results of each evaluation are described next.

5.1 Student usability study

The aim of the student usability study was to identify the strengths, weaknesses and possible improvements that would make Submit! a more user-friendly system for students. In identifying the strengths and weaknesses we were able to establish the level of intuitiveness, that is, whether a student could easily figure out the steps to complete a task without assistance by persons or by help. Finally, we were able to identify some useful features whose implementation will improve Submit!. The evaluation consisted of heuristic evaluation, observation of students performing given tasks and a questionnaire.

5.1.1 Heuristic Evaluation Performed by the Investigators

Heuristic evaluation is a simple and inexpensive technique that allows systematic assessment of a product to identify potential usability problems. The heuristics used in this study include:

- Simple and Natural Dialogue
- User oriented language
- Users memory load required
- Consistency across interfaces
- System feedback following user actions
- Clearly marked exits
- Shortcuts
- Appropriate error messages
- System error prevention
- Help documentation

Prior to using the technique specific questions are posed. For example to determine if the dialogue is simple and natural the investigators asked the following questions for every screen that is part of Submit!.

- Is there any information that isn't needed ?
- Is there anything pertaining to dates that may become outdated?
- Is information in logical order ?
- How does the dialogue affect the visibility of the interface?

Based on the heuristic evaluation changes were made to the interface involving changing links, buttons and labels.

5.1.2 Usability Evaluation Performed by Student Participants

After heuristic evaluation by the investigators a usability study was conducted followed up by surveys to the participants. The integrated questionnaire and task list aimed to have the participants use most student aspects of the Submit! system. Once the evaluation was complete we could access a log transcript of the participants evaluation session. This transcript allowed us to cross- reference the tasks or actions performed by the participants to see whether they were performed successfully, in the correct order and on their first attempt. There was no conflict between the log transcript and the answers entered by the students in the questionnaire. In analysing each questionnaire, we concentrated on the following items:

- Percentage of participant's that successfully completed the fundamental tasks.
- Number of errors made by each participant.
- Number of system errors.
- Participant's awareness of and the usefulness of help.
- Number of times the participant got stuck and requested the observers' help.
- Specific problem areas.
- Individual comments made by the evaluators regarding the overall system appeal and speed.

Participation was on a purely voluntary basis. We used 1st, 2nd and 3rd year students from the Department of Computing. The three years aimed to represent amateur, intermediate and advanced users respectively. Most 2nd and 3rd year evaluators had already used Submit!. The 1st year students had not used it. The results are shown in the next subsection.

5.1.3 Results

The survey involved 39 participants (34 male and 5 female), aged between 19 and 53. 87% had used Submit! before.

The obvious weak areas identified were:

- Students not knowing the difference between downloading and deleting.
- Lack of awareness of the bug report.
- Unintuitive menu style.
- Dislike of some aspects of the screens' appearance.
- Slowness of the system at times.

5.2 Academic usability study

This study focused on the aspects of the system used by markers and lecturers. These users have entirely different objectives compared to students. Their main concern can range from simply downloading a zip file of students assignments for marking, to the administration of assignments and subjects, whereas students are mainly concerned with simple submissions. The screen shot in Figure 1 shows some of the tasks an academic can perform using the system.

A number of instruments were used in this study including: surveys, thinking aloud, heuristic evaluation and comparison with "rival" systems from other universities.

Tasks that were assessed:

- Basic Assignment Creation (see Figure 2)
- Advanced Assignment Creation -this includes the restriction of submissions by file name, by file size, and by enrolment, selected students and date.
- Modification of the Assignment Parameters
- Automatic Testing of Submissions
- Reporting – allowing lecturers to observe a listing of students and their results for the current assignment, sortable by result and other criteria such as first / family name.

The thinking aloud method is used for assessing how the user views the system and what misconceptions they might hold. We combined the technique with a form of cognitive walkthrough to see if the user's experience and the information provided would lead to the next correct action. The process involved taking a user through a series of eight tasks and simulating the users problem-solving process at each step in the users "dialog" with the system. For each of the tasks the investigator asked:

- Was the correct action sufficiently evident to the participant?
- Was the correct action noticed when it was available? For example, was the help button visible when needed?
- Did the user know from the feedback that actions where successful?

We used a first time user for the walkthrough/thinking aloud. This user successfully performed all tasks so we did not conduct a further test with an expert user. Had time and subjects been available, it would have been good to repeat the walkthrough with one or more additional novices to ensure that our first participant was representative of the novice population.

From the results of the walkthrough, heuristic evaluation and discussion with the designers of the system a questionnaire was formed. The questionnaire starts off by asking some basic demographic information on the topic of user experience. The next range of questions are fairly general, the aim here is to focus the users attention, hence encouraging a longer and more personal response. The remaining questions are closely related to usability heuristics.

25 questionnaires were distributed to staff via pidgeon holes. By the final date of the study 15 questionnaires had been collected. 13 were completed and returned.

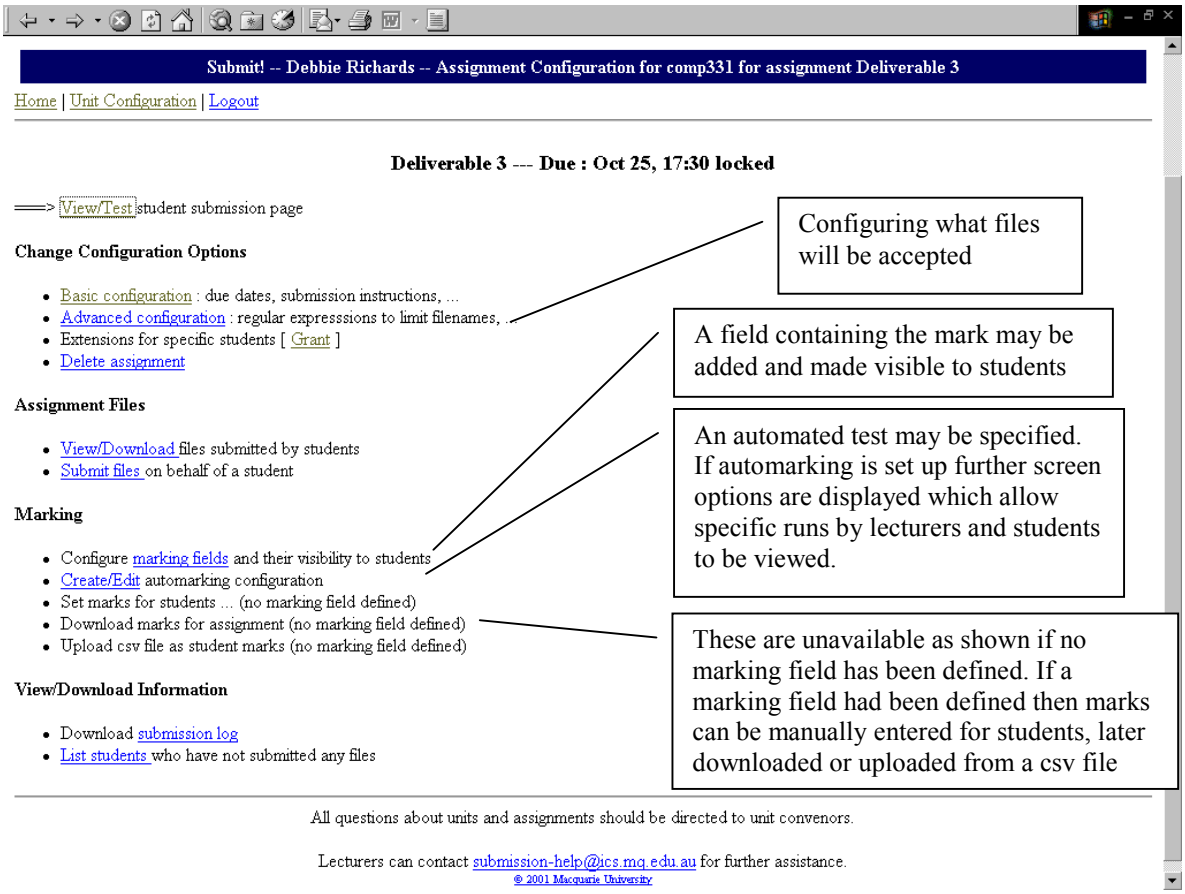

Figure 1: Assignment Configuration Screen

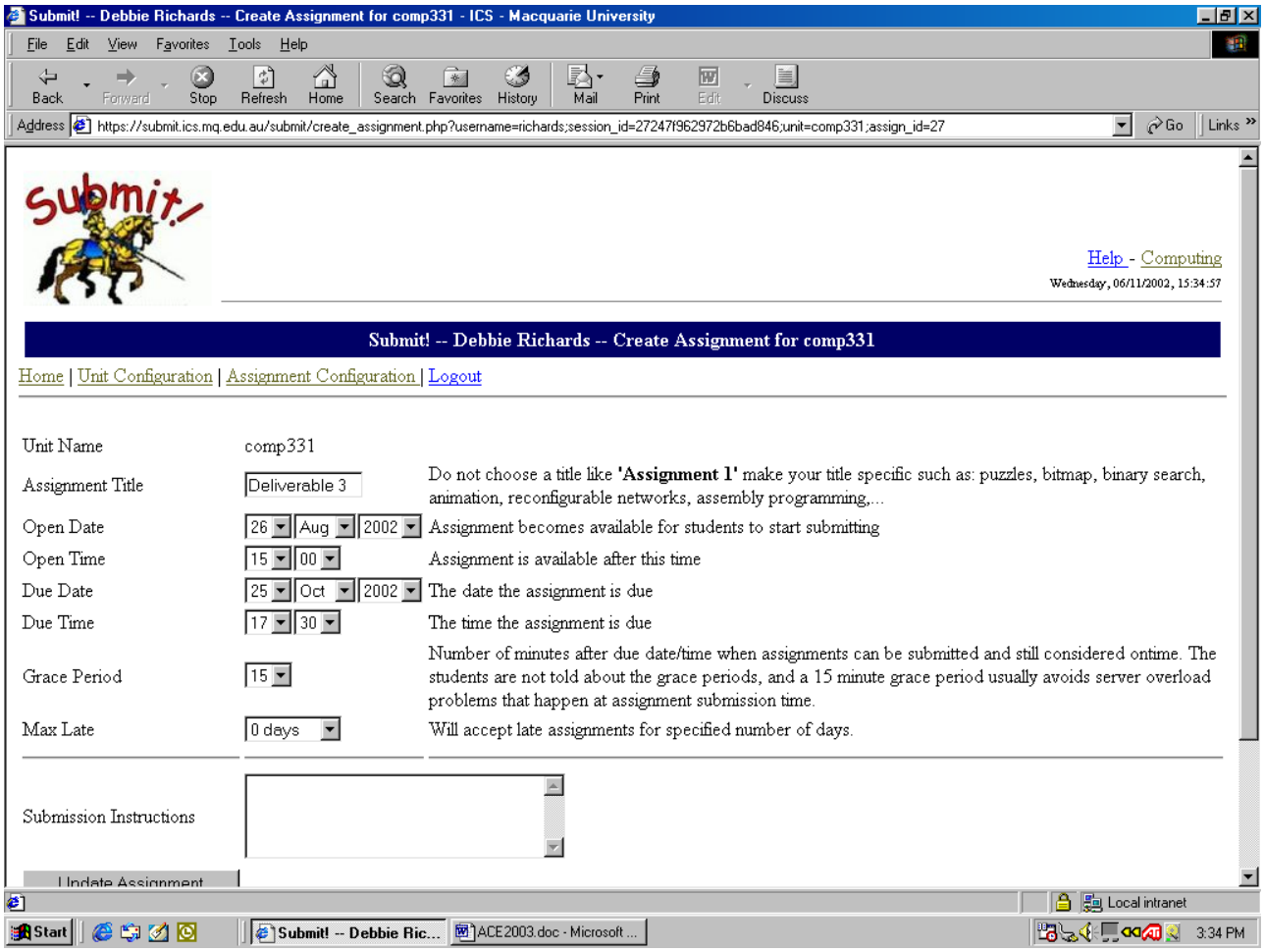

Figure 2: Basic Configuration Screen

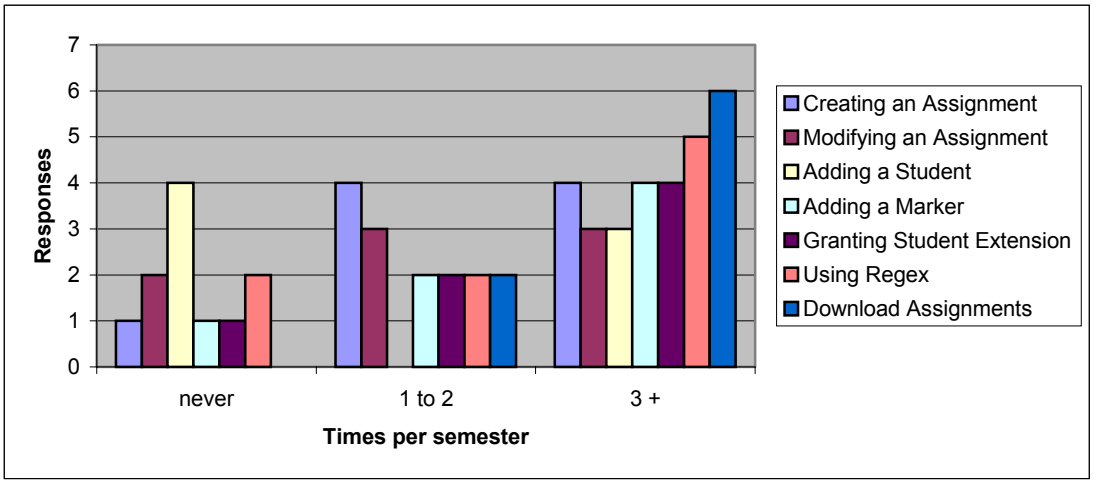

Figure 3: Planned feature usage

5.2.1 Experience (Question 1 – 2)

All respondents had used the submission system. The results show that the current most used tasks are Downloading Assignments (this might be due to the influence of a few markers, however this is still very high as the total number of responses for the task is the lowest, seven) followed by Creating Assignments and Modifying Assignments.

It is worth noting that there is a considerable rise in the planned usage of features such as using regular expressions to specify filenames. Planned usage is given in Figure 3.

Five of the respondents didn't use help and the other six used help one or two times. No respondent used help more than two times. The high number of people that had not needed help tends to indicate that the system is very intuitive.

5.2.2 Question 3

This question related to the use of Submit!'s online help. Five of the respondents didn't used help and the other six used help one or two times. No respondent used help more than two times. The high number of people that had not needed help tends to indicate that the system is very intuitive.

5.2.3 Question 4

Eight people said that Submit! was intuitive to use when they first started. Some problems where:

- The system was hard to navigate.
- The interface is easy until you run into trouble.
- Users didn't realise that you needed to "show all" before you "selected all."
- Too many options, wasn't sure what to use.
- "Show all students" isn't immediately obvious.

5.2.4 Question 5

Six people said that submit lacked features they believed it should have. Examples of such features were:

- A reporting feature to show all students and their marks on the *same* screen.
- Better marking interface.
- Enhanced support for hand marking.
- Results available for students.
- Ability to return all comments, etc. to students in paperless mode. This requires a file annotator.

5.2.5 Question 6

Four people identified problems that could not be solved with Submit! Examples of such problems were:

- An inability to detect and impose restrictions on executables and word documents. e.g. one student was able to submit an executable called Makefile.
- Auto marking has many associated unpredictable outcomes ("nastiness").
- Another lecturer mentioned that CSV files were unable to be uploaded.
- There were also concerns regarding the handling of Grace Periods, one lecturer explained in the hallway that users were told submissions were closed during the grace period, perhaps they should be told this when the grace period is over (otherwise there is no point having the grace period). This would also make sense, as students don't know when the grace period ends, and won't abuse it.

As mentioned above heuristic evaluation was also conducted by the usability engineer which critically assessed the Submit! System according to the usability heuristics in sections 5.2.6 - 5.2.15.

The results of the survey, the walkthrough, and a basic Heuristic Analysis have been summarised below. In analysing the questionnaire, a direct translation of results from the "Strongly Disagree" to "Strongly Agree" scale has been converted to a numeric value between one and five. A rating of one indicates poor interface design and a value of five indicated very good design.

5.2.6 Consistency

People like to feel that they understand a system and can predict its behaviour. Consistency is about making the system perform the same way for every action. It is also

about consistent language. The participants of the questionnaire gave the system an average rating of 3.72 and 3.63 for consistency. A couple of inconsistencies were noted such as a "Configure Assignment" section under "Assignment Configuration". One should not be part of the other or should be given a new name.

5.2.7 Simple and Natural Dialogue

Simple and Natural Dialogue describes having an interface with only necessary information and no clutter. The aim of this heuristic is to measure how cluttered an interface is and also how easy information is to find. The participants of the questionnaire gave the system an average rating of 3.3 and 3.4 for Simple and Natural Design. The users felt the interface had a standard level of simplicity. The system did have a problem when it came to getting marks. Rather than being able to view all marks at once, the lecturer had to individually look at marks, which is very tedious. Perhaps more description could accompany each feature. One user said, "Lot's of options – wasn't sure what I needed to use."

5.2.8 Speak the users language

An interface should be in the users native tongue and avoid using jargon and "industry words" such as L317 for a currency type, it should use clear language that users understand like "British Pound." The responses averaged at around 4 (agree) with a standard deviation of 1. So we can conclude that users generally found the system easy to understand. During the Heuristic Analysis it was noted that the following changes might improve the interface. The "Advanced Configuration" link should be changed to "Limit filenames" since the link is only used for changing filenames. Also change "Extensions for specific students [Grant]" to "Grant Student Extension." A way must be provided for having the features of a regex, without the need for learning regex's. This is a matter of concern to some users. No matter how simple regex's are, there should be perhaps a tick box that says only this name allowed (i.e. prefixes the '^' and affixes the '$' and escapes all other characters). From analysis of competitor software we found Submit! better and than the Comp.Eng. submission system at UNSW (students had problems submitting based on the regex implementation that all/non of the files be submitted from prompt at once).

5.2.9 Provide clear and adequate Feedback

Providing clear and adequate feedback is a question of both the speed at which the feedback is displayed, wether the system status is visible and the quality of the feedback. Users agreed that the system kept them informed but were not so pleased with the response times citing examples such as taking 20-30 mins to create zip files for four hundred second year programs. Heuristic evaluation also revealed that instead of saying "assignment X has successfully been changed" the system either shows the old and new value pairs, or the entire updated record. The system could also be expanded to check file types, binary and word files for example.

5.2.10 Minimise Memory Load

Minimising memory load is a heuristic describing the amount of information that must be transferred between dialogues and how information is stored within dialogues. Information on a dialog should include predefined values and instructions for the task to be completed with the screen. A user should not have to memorise a sequence of steps or continuously refer to a manual. Five of the respondents hadn't used help and the other six used help one or two times. No respondent used help more than two times which suggests that the user didn't have to keep referring to help so the interface was good in this regard. Respondents agreed with the statement stating that they didn't have to remember instructions.

5.2.11 Good Error Messages

Good error messages refer to having error messages that are understandable, clearly describe the problem in concise non-technical terms and describing in steps the way of avoiding the error and fixing the problem. The users didn't agree that the system gave appropriate error messages with a score of 3.18 being slightly better than neutral but one of the worst overall marks. Furthermore the results for the number of help messages and their helpfulness are all just slightly above neutral.

5.2.12 Prevent Errors Occurring

Preventing errors from occurring can be done in a number of ways. The system designer can place clear and consistent instructions and provide restricted input fields such as integer values between 1 and 31 for days of the month. When input ranges exceed the expected the user was informed immediately rather than having something go wrong later. Another method is to avoid text fields or any place where the user might be able to type and incorrect value. The Submit system did well for most parts in restricting inputs, especially for dates. Though setting up defaults for dates often caused them to be overlooked creating more errors. Users found that they made mistakes when initially using the system. However these errors don't reflect too poorly on this heuristic as they rarely make the same mistake twice. People have however explained difficulties with features such as regex's. Something that can be easily solved with concise guide containing examples and steps to designing regex's.

5.2.13 Shortcuts and Accelerators

Shortcuts and Accelerators provide a means for users to improve the speed repetitive tasks. Examples may be having two separate system interaction methods for beginners and experts (for example having a mouse and a control key to activate the same button). Another example is to provide templates. Navigation is an area that Submit has problems. Perhaps the interface should be expanded to have a greater depth in menus (like a side bar for adding students, updating assignments, etc.) the user should also be returned to the pages the page they came from after performing a task, not just sent to the Unit Configuration page. In addition at the top of every page there should be an index so the user can jump to any appropriate section, and likewise a link back to top after each section. A less that neutral mark was achieved for

this feature. However Submit does provide accelerators where it is important, it provides mechanism for automating assignment marking which save users a great deal of effort. It could be improved to allow templates to be made for weekly assignments, or shortcuts to automate the process of making a weekly assignment.

5.2.14 Help and Documentation

Help and documentation describes how adequate the help is. Is the help searchable on task? Is the help organised based on task and provide clear concise and simple steps to solve the problem (complete the task). Submit fails on all of these measures. It did no better with the manual. Users found help outdated or missing. In addition users had problems finding the help as it quickly disappears when the user scrolls. This report suggests a separate frame at the top providing navigation details and help. In addition, as a quick suggestion for improving the online help, would like to see each heading be a reference to an html anchor on http://www.comp.mq.edu.au/~apce/ manual/ SubmitInstGuide-v1.htm relating to the heading.

5.2.15 Clearly visible exits

Clearly visible exits can prevent the user from feeling confined in a task. If the user makes a mistake, can they go back or cancel the dialogue. Is the site easy to navigate? Can the user quickly exit the system if they need to? According to the survey the users had little problem with exits. This is the case for most web-based systems though. One thing this system lacked is the ability to undo tasks like "deleting files" perhaps this could be implemented in the future.

In summary, Sumbit! performed very well in the walkthrough/thinking aloud and participants gave it better than neutral ratings for most of the Likert statements in the questionnaire. As a result of the questionnaire and the Heuristic Analysis, many suggestions have been made and usability problems identified.

5.3 Evaluation of Impact of Submit! on Student Results

We have conducted an initial study in one large first year programming unit. The study uses data over two semesters: S1 2002 and S2 2002. In Semester 2 we have incorporated programming style checking scripts into Submit!. Students are able to submit their assignments as many times as they wish prior to the assignment due date and receive feedback on their programming style. Currently only following guidelines are being checked:

Too Long: All program lines should be less than 78 characters.
Tabs: Should use space instead of tabs.
Paren_Curly: There must be space left between a closing parenthesis ')' and a an opening curly brace '{'.
Empty lines: Do not have more than five empty lines in code.
Code_Curly: Do not have code on the same line following a curly brace '{'.
Block_Single: Do not use a whole block in a single line (an opening curly brace '{', followed by code, followed by closing curly brace '}]).
Comment space: There should be space between the indicator for comments (usually '//' or '/*') and the actual comments.

These guidelines are a small subset of our complete *ideal* programming style. The guidelines were chosen based on our perception of most common problems in student code and the ease of implementing the automated tests. The "empty lines" criterion was later eliminated since no student had this problem in their code.

In Semester 1 students used Submit! to submit their assignments, but could not run the automated tests to get feedback on their style problems. This semester we have made automated testing available. At this stage we have only collected data for the final submissions for assignment one, shown in Table 1. The students show slight improvement in most of the categories, except for the "too long" category. One possible explanation for this is that unlike the assignment in Semester 1, the assignment in Semester 2 required students to print multiple lines of text.

Criteria	S1 no feedback	S2 feedback
Number of Assignments	232	261
Too Long	29%	31%
Tabs	92%	14%
Paren_Curly	24%	14%
Code_Curly	24%	13%
Block_Single	22%	8%
Comment Space	10%	7%

Table 1: Percentage of students who had a style error with and without feedback prior to final submission

We found that only 77% of the students ran the automated test. We attribute this to either students not being aware of this new facility or to students submitting the assignments right before the deadline leaving no time to fix any mistakes.

There was no limit on how many times students could run the automated style checker. We expected that some students would eliminate all their errors by repeatedly running the style checker. Some students ran the checker over 10 times. We were interested to see if there was any trend on the amount of usage and the assignment results. To check this we grouped the data according to usage categories (once, 2-5 times, more than 5 times) and analysed the number of errors for each group.

Figure 4 below shows the general trend that students who ran the style checker more times have fewer errors. We noticed that students who ran the test once had slightly higher error rates on a number of indicators. There are two possible explanations for this effect:

1. The style checker takes anywhere from two to ten minutes to complete. For students who ran the style checker right before the assignment deadline, by the

time the style checker was complete the assignment would already be locked preventing students from fixing their mistakes.

2. In addition to the style checker, there was also a basic input-output checker that was available to the students. Students who ran both the input-output test and style checker may have decided to fix mistakes related to input-output rather than style errors. We are currently trying to test these hypotheses by talking to students and examining system logs.

While having a two to ten minute delay in getting automated test results is unacceptable in terms of usability, it prevents students from repeatedly running the automated tester and only fixing the problems the automated tester points out. Testing a program is a valuable and necessary skill. We are considering limiting the number of times a student can run an automated test in order to avoid having students become too reliant on the automated tester.

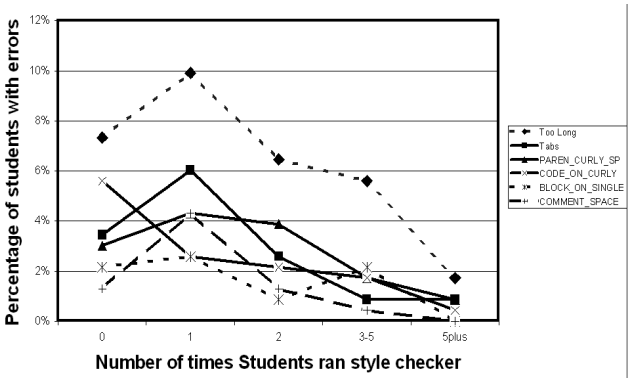

Figure 4: Error trends based on how many times students ran style checker

6 Conclusion and Future Work

So far the Submit! project has met its goals in providing a consistent and effective framework in which feedback can be given to students on their programming work. Our incremental approach has enabled even early versions of the system to help students and teachers get their work done more effectively.

Our evaluations have shown that the system is generally usable while pin-pointing areas of improvement that we are addressing in current development. These studies have concerned the more superficial aspects of the system – namely usability of the tool. However, a preliminary comparative study of student assignment performance has shown that providing feedback for self-assessment can provide benefits.

In the future, we would like to capture and encode the knowledge we gain about useful forms of critiquing so that it can be reused. In particular, we intend to create a library of reusable modules to enable similar feedback for programs written in different programming languages. In developing the libraries, we will discuss with students and staff how feedback should be structured such that the feedback is informative and thought-provoking for the students without making them reliant on the system.

We will also incorporate plagiarism detection to check for similarities between submissions and notify staff of any suspicious cases.

As a longer term goal, we will explore how we can support critiquing for other types of submissions. For example, in some of our units students are required to submit designs rather than actual program code. The designs are expressed using formal graphical notations, so we should be able to convert them to text for analysis. Any discipline that makes similar use of structured submissions will be able to adapt our tools for use in their units.

7 Acknowledgements

We are grateful to our university for funding to support this project under the Macquarie University Flagship Teaching Development Grant scheme. We also acknowledge the significant assistance and feedback provided by academic, student, administrative and technical support members of the department and division.

8 References

ANDERSON, J. R., and REISER, B. J. (1985) The LISP tutor. *Byte*, 10, pages 159–175.

ANDERSON, J. R., BOYLE, C. F., and YOST, G. (1986) The geometry tutor. *The Journal of Mathematical Behavior*, pages 5–20.

ARNOW, D. and BARSHAY, O. (1999) On-line programming examinations using WebToTeach. In *Proceedings of Innovation and Technology in Computer Science Education*, Cracow, Poland, pages 21–24.

BRIDGEMAN, S., GOODRICH, M. T., KOBOUROV, S. G. and TAMASSIA, R. (2000) PILOT: An interactive tool for learning and grading. In *Proceedings of the 31st ACM SIGCSE Technical Symposium*, Austin, Texas, pages 139–143.

BROWN, J. S., BURTON, R. R., and DeKLEER, J. (1982) Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III. Pages 227–282 in D. Sleeman and J. S. Brown (Eds.), *Intelligent Tutoring Systems*. New York: Academic Press.

GIBBS, G. (ed) (1994) *Improving Student Learning — Theory and Practice*. Oxford: Oxford Centre for Staff Development.

HATTIE, J.A., JAEGER, R.M., and BOND, L. (1999) Pervasive problems in educational measurement. *Review of Research in Education*.

LAJOIE S.P. and LESGOLD A. (1989) Apprenticeship training in the workplace: computer-coached practice environment as a new form of apprenticeship. *Machine Mediated Learning*, 3(1), pages 7–28.

RAMSDEN, P. (ed) (1988) *Improving Learning: New Perspectives*. London: Kogan Page.