Mass Production of Individual Feedback

A thesis submitted

in the subject of

Computer Science

for the degree of

Master of Science

under the supervision of

Charlie Daly

for

Dublin City University.

By

David Heaney

July 2006

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of M. Sc. in Computing (insert title of degree for which registered) is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: [illegible]

(Candidate) ID No.: 96375001

Date: 19/07/'06

Contents

List of Tables

List of Figures

Abstract

Learning to program is intrinsically difficult. In addition there is a trend towards increased student diversity and larger class sizes. Student diversity increases the need for individual attention for each student, while increased class sizes decreases the amount of time a lecturer has to provide this attention.

This thesis investigates an approach to help provide each student with detailed individual feedback. This feedback is important where individual attention is lacking. We used two trials to determine the effectiveness of the system.

The first of these trial runs was in Autumn 2002 and the system provided the facility to the tutors to give personal and detailed feedback to each student. There was a statistically significant improvement in the weaker students' exam results for that Semester.

The system was improved for the second trial. The tutors could provide feedback as before, but also this time the students could provide feedback on each other through peer-assessment and self-assessment. The system remained popular and useful. However for the second trial we were interested in making the peer assessment aspect a success. The thesis will discuss our limited success in this area. Eighty eight percent of the students were motivated to provide feedback through bonus marks and the incentive of getting feedback from others, but many students gave absolute minimum feedback.

Acknowledgements

First I have to thank my supervisor Charlie Daly for giving me the opportunity to pursue a Masters and in having the confidence in me to allow me to run my system on his Introductory Programming module. On the last draft of the thesis I considered taking this acknowledgement out. However on further reflection I decided that Charlie also deserves special thanks for driving me so hard right to the end and helping me to realise if you want something that bad you sometimes have to depend on yourself! Also thanks to all the students who used the system, especially to the tutors, who really made the system work. Without them providing the feedback it would have been nothing. Thanks also to all their suggestions on improvements, and thanks for their eagerness to point out bugs in the system! They kept me on my toes.

Finally thanks to my friends and family for all the support and inspiration that they have given me over the last while. My Dad has always been there for me when I needed and has always given me my own space to develop when that was needed too. All my brothers (Joe, Stephen and Christopher) and my sister (Therese) have all their own ways of making sure I stay grounded and true to myself. My eldest brother Joe has always been a leader to me; someone I looked up to and I have always tried to follow in his footsteps. At the moment that is not the case but again, in time, it will be. There should also be a special mention for my mother. She has given us everything. Everything I have is in some way down to her sacrifices. These two people in particular have been instrumental in getting me to where I am today.

Chapter 1: Introduction

Today there is a great demand for people with Computing and Programming skills. The task of producing computer literate graduates ready for today's increasingly high tech industries belongs to our modern Universities. Programming is a skill and cannot be acquired by rote learning. It requires a true search for understanding. The challenge for Universities is to provide an environment that encourages the students to take on this quest for understanding. Evidence shows that they are failing to do this [37], [40].

In 2001 a report [40] commissioned by the Higher Education Authority (HEA) in Ireland identified Computer Science as the area with the highest non-completion rate in Irish universities. The report showed that 26.9% of the students embarking on a Computer Science course failed to graduate. This compares unfavourably with the national average for all areas of study, which was 16.8%. The unique factor of Computer Science courses is the requirement of learning computer programming. This suggests that there are problems in Programming courses. This evidence is also backed up by an international study [37] of programming skills of computer science students. The study concluded that most students, after taking a course in Introductory Programming, could not program at a satisfactory level.

This thesis proposes some reasons why so many students drop out or perform poorly in computing courses – large class sizes, student diversity and lack of individual instruction. An online administration system to manage individual feedback during a programming course was developed to address these problems. The main focus of the thesis is to carry out an evaluation of this system.

Many researchers have already looked at the problem of teaching Introductory Programming skills. Lawheard of Mississippi University has done research [32] on using Lego Mindstorm kits and Lejos (java for Mindstorm) to design assignments for Introductory Programming modules. Researchers at Duke University have experimented with animation and virtual worlds to introduce their students to Computer Science [51]. Among the tools that they have used are JAWAA (a java tool for creating animations), Starlogo (a programmable modelling environment), Alice (a 3D interactive programming environment) and Karel++ (a tool for programming robots which move graphically in 2D worlds).

These efforts at making computer programming more interesting are commendable, but we might make more progress if we can find the root of the problem. Then use a direct approach to solve it.

The growth of high tech industries means more students are taking Computer Science courses. In addition students in Engineering, Sciences and Maths are studying programming. Inevitably this means class sizes increase – and students get less individual attention. With the traditional lecture structure and assignment submission systems it is increasingly difficult to accommodate for large numbers. Assignment correction involves managing hundreds of scripts, floppy disks or email attachments. This makes it extremely difficult for a single lecturer to handle the workload.

In addition to the problem of reduced individual attention and administration overload, large class sizes also result in greater student diversity. Clearly since students from engineering degrees and science degrees are now also taking computer science courses, the classes tend to be more diverse. Students have varied experience and different expectations. With the extra diversity there is a danger of the students falling into two categories:

1) Those that believe that they will not be able to learn this new skill, which may seem strange to them. A lot of students taking Introductory Programming courses have never programmed before or have had very little experience with programming. In the past this was not the case.

and

2) Those that do not value the outcome. The student that does not plan to pursue a career as a developer or software engineer will not value the module. Many of the students will be hoping for careers in other fields of computing or in a different field of science altogether.

Students that fit into either of these categories will be poorly motivated to learn to program. The implications of this are discussed by Jenkins in his paper "Teaching Programming – A Journey from Teacher to Motivator" [25]. In respect to those that believe that they will not be able to learn this new skill; he advises that it would be a lie to tell them that programming is easy to learn but they must be reassured and supported, and they must be made to expect to succeed. He also addresses the second problem, urging that the students must be made value the outcome, rather than the grade. The problem can be neatly summarised into a simple equation.

$$\text{Motivation} = \text{Expectancy} \times \text{Value}.$$

Expectancy and value are multiplied; therefore to keep student motivation high, both factors need to be considered.

This problem of motivation can be addressed using appropriate feedback. In his ARCS Model of Instruction [27] (discussed in more detail in the Literature Review), Keller identifies feedback as a means to develop a student's confidence and satisfaction. A more confident student will expect to succeed in learning. Also the students value satisfaction or enjoyment, if they enjoy programming they are more likely to be motivated to learn it. So with the right feedback we can keep both factors in our motivation equation high.

The timeliness of this feedback is also important. Keller's Motivational Delivery Checklist [28] says that a course that "provides feedback on performance promptly" will increase the value of the course. Race [49] argues that "the greater the amount of feedback that learners receive before the end of course assessment, the greater their opportunity to learn from such feedback".

When students get timely, quality feedback they will be more motivated to learn.

There is a real problem with providing this feedback, especially for large classes of students. It would be unreasonable to expect a single lecturer to carry out this task on an ongoing basis. A better solution to this problem might be some automated process. However, the feedback must be personalised and tap into the individual motivating factors that each student has. Also, there needs to be less emphasis on whether code works or not (a feature of automated marking) and more emphasis on good programming practice (such as, when to use a for loop, as opposed to a while loop, or drawing attention to good or bad variable and method names etc.). The solution we found was to develop a system that would delegate the marking of assignments to a group of tutors. The system would minimise their workload by taking care of all administration issues, and it would maximise their efficiency by providing an easy-to-use and powerful user interface.

In previous work carried out in DCU [15], a tutor system was put in place where the second year students supervised and tutored the first year students' lab sessions. This was successful. But it was found that the students just did not receive adequate or timely feedback on their assignments from their lecturers. We took advantage of the structures already in place, and introduced our system into this environment.

The first trial of the new system was in the Autumn Semester in 2002. This trial revealed that feedback is indeed an integral part of learning. The grades of the weaker students improved by small but statistically significant levels. All the users of the system enjoyed using it and found it helpful.

The second run of the system was in the following Autumn in 2003. This time there were a few added features, including self-assessment and peer-assessment, and an online Java Virtual Machine so that the tutors could run the assignments. Peer-assessment and self-assessment will increase the amount of feedback that the students get. It will have other benefits too. Once again the users found the system helpful and enjoyed using it. Login sessions in the evenings and the weekends were proving that students were getting quicker access to feedback than before. We also successfully motivated the students to provide feedback to their peers, with 88% of students completing the assessment process every week.

Chapter 2: Literature Review

Introduction

The key areas that will be addressed in the literature review can be broken down into two categories – educational theories and feedback technology. From an educational point of view the focus is on feedback. Feedback can help student motivation. Feedback can motivate deep learning. Students can get feedback through peer-assessment and self-assessment. In paired programming students are constantly getting feedback from their partner. All these ideas are covered in the literature review. This background makes it possible to understand how the students can benefit from a system that gives timely individual feedback to a large group of students. It also shows various techniques used for providing this feedback. Finally there is a section describing some research projects that used technology to provide feedback. These projects have been mostly successful. However they can be improved by a system that would emphasise more of the accepted educational theories discussed in the chapter.

Motivation

For learners, motivation can be classified into intrinsic motivation and extrinsic motivation. Extrinsic motivation is when the learner is motivated by external factors, such as grades. Intrinsic motivation is when the learner is motivated by their internal desire to learn or because they believe that it is the good or right thing to do [33]. Lepper's research on motivation favours intrinsic factors over extrinsic because the goal is the learning rather than the reward.

Jenkins discussed motivation in his paper, “A Journey from Teacher to Motivator” [25]. He maintains that expectancy and value are important for motivating learners. When a student does not believe that he can master the material then he will be poorly motivated. Likewise, if the student does not value the outcome (either in an intrinsic sense, or an extrinsic sense) he will be poorly motivated. It is important to see that the student must both value the course and expect to succeed.

Keller [27] summarises motivation with the following 4 points:
Attention – The student’s attention must be on the lesson.
Relevance – This involves showing the learner that what they are learning is useful.
Confidence – Confidence is increased by making the expectations for learning clear to the student, by giving the learner ample chance to succeed and by giving the learner personal control.
Satisfaction – Making the learning experience as interesting as possible increases satisfaction.

These ideas of Jenkins [25], Lepper [33] and Keller [27] were incorporated into the feedback system to help motivate the students.

Deep Learning

Deep learning occurs when students try to deconstruct and reconstruct knowledge so that they understand it. Surface learning occurs when students try to remember as much as possible from a text [53].

Many students can rely heavily on surface learning and may have picked up the habit of rote learning in secondary school. In particular the science curriculum for the Leaving Certificate course "promotes rote learning and recall of scientific facts, with insufficient emphasis on building higher order skills" [56]. It is easy to see more general examples from right across the curriculum, for example the student must memorise dates for History; poetry and quotes for English; conjugations for languages. This approach to learning will not work when applied to Computer Programming. In Programming courses, the students will need to come up with an original analysis of their own in order to complete an assignment. "Programming is not a single skill but a multi-layered hierarchy of skills" [4]. Original ideas cannot be learnt by rote. Deep learning will help the students be able to understand the concepts taught in the module in more depth. They will be able to construct their own original ideas.

Peer assessment and self assessment have been put forward as ways of encouraging students to reflect more over their work [55], [57] and can thus help to develop deep learning.

Feedback

Feedback is an integral part of the learning process. Feedback is needed both to help a student learn from mistakes and to provide encouragement.

Race [49] writes that the timeliness of feedback is paramount. Research has shown that immediate feedback is more advantageous than delayed feedback [1]. "Applied studies using actual classroom quizzes and real learning materials found immediate rather than delayed feedback to be more effective" [31]. When feedback is received soon after the assignment is completed then the ideas are still fresh in the student's mind. Any corrections or comments on those ideas will be better understood.

Kolb [29] attempts to model the learning process. He uses a circle (see figure 1) to convey the idea that the learning process does not end with the doing of a task. Rather, the learner immerses himself or herself in the task. Then the learner reflects on what he or she did. Then s/he conceptualises the task by getting a better understanding of what s/he did, why s/he did it and why it worked. The final stage is the planning stage where the learner will consider how he can do things differently the next time. So, the learner is back at the beginning of the circle again. What he or she has learnt from the first assignment will be fed into how they approach the next assignment. The instructor can help to guide the learner through the reflection, conceptualisation and planning stages. With no feedback there is a danger that the learning ends with the doing of the assignment. The student may not take the time to reflect and plan without the guidance and encouragement of the tutor's feedback.

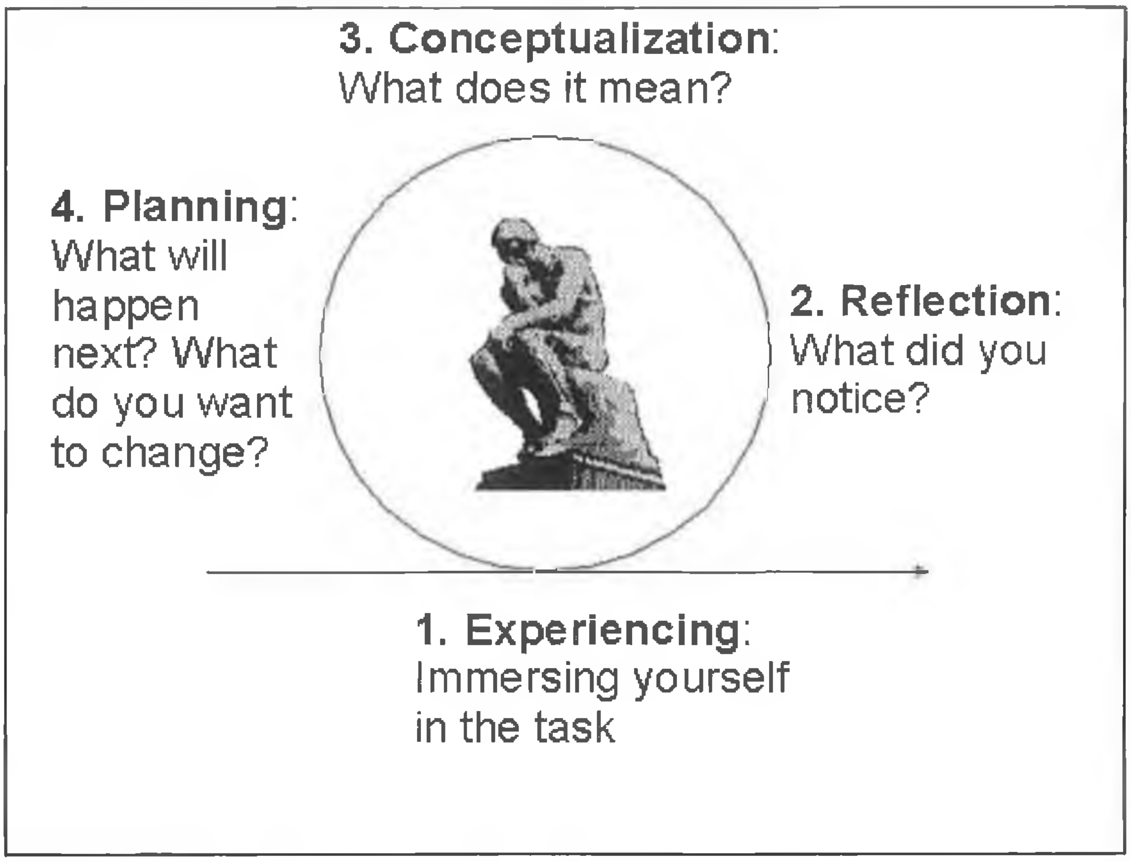

3. Conceptualization: What does it mean?
4. Planning: What will happen next? What do you want to change?
2. Reflection: What did you notice?
1. Experiencing: Immersing yourself in the task

Figure 1: Graphical representation of Kolb's learning cycle (the famous Thinker Statue by Auguste Rodin is depicted in the centre of the cycle).

Individual Instruction

Socrates took a one-on-one approach to teaching his students. Xenophon writes that Socrates sought "very carefully to discover what each of his companions knew. Whatever was appropriate for a gentleman to know he taught most eagerly" [6].

Good feedback must be personalised since for each student the main factor for valuing the course is unique [25]. Many students will value the end of year marks, as it will mean progress for them to the next stage of their education. It should not be forgotten, however, that receiving encouraging feedback and getting credit for their efforts, are all valued by students. Some students are motivated by a desire to please their tutor or their lecturer. Often classes are too large for one individual to supply personal feedback to each student.

On the problems with automated feedback Murray [41] writes: "The problem of detecting and correcting errors in programs is difficult because of significant variability". In other words it is difficult for an automated system to correct an assignment. Oderkirk-Hash [46] and others have had limited success with automated systems. The InSTEP system developed by Odekirk-Hash caters for a simple assignment. They concluded that the system could be used side-by-side tutor feedback, however for weaker students in particular advised that tutor feedback is still important.

Another advantage that personal feedback has over automated feedback is the human ability to recognise good style. There is a matter of style to good programming [58]. A program might work, but be badly written. Automated marking can provide feedback quickly but cannot tell if the code is well designed. (There has been work done in this area [47] but since good programming style means that a human should understand the code there is an inevitable human link [21]). Students must learn to write code that can be easily understood and modified by another person. This is a key skill for a programmer. Through detailed feedback the student can learn these skills from their assignment.

Peer Assessment

Peer assessment [9] is defined as the process of having the students mark or assess one (or more) of another student's assignments. Peer assessment provides the following benefits for the learner.

1. The student can learn from other students' mistakes as well as their own [8].
2. The student gets extra feedback from someone other than themselves without putting an extra burden on the tutors [17].
3. It takes the burden of providing feedback off the tutor. The added burden on the students themselves is justified as it improves their programming ability. Evaluation is the highest level of Bloom's taxonomy of education [11].
4. The student begins to understand the assessment process more clearly after peer assessing [12].

A study in Dublin City University (DCU) used a near peer assessment method [15] where the second year students that received the top marks in their first year programming exam supervised the new first year students. The first year students did not get the benefit of assessing their peers. They did however get more tutor time than would have been otherwise possible and they received help from people who had just completed the same course of study. For the second year tutors it was useful as revision.

Self Assessment

Self-assessment is a process where the creator of a piece of work is responsible for assessing it. Usually the teacher will set the criteria [16]. Like peer assessment it is used to encourage the students to look more deeply at their work and to give them a better understanding of the assessment process [48].

Self-assessment works well when the student takes an "outsider's" point of view to evaluate his or her own work. The problem is that it is often very difficult to look objectively at something that you have created yourself. We asked the students to peer assess another student's work before they were presented with their own work to assess. This should get them into the right frame of mind, i.e. it gets them thinking critically.

"For with what judgment you judge, you will be judged, and with the measure you use, it will be measured back to you. And why do you look at the speck in your brother's eye, but do not consider the plank in your own eye?" [36] So, even at the dawn of the Christian era it was believed that examining the specks in other people's code could help them remove the planks from their own.

Pair Programming

Pair programming is a method where two programmers work on one computer. Partners use their combined intellect to develop the program. One partner types (the driver), and the other (the navigator) checks as he types. They switched roles regularly. It was developed in industry as part of the eXtreme Programming (or XP) philosophy [18], [24]. XP has been generally accepted throughout industry as a means of decreasing development time and producing more robust software. It achieves this by highlighting the planning game, small releases, simple design, testing, continuous integration, refactoring, pair programming, collective ownership, 40-hour week, on-site customer, metaphor and coding standards. Kent Beck pioneered it in 1996 for Chrysler's payroll system, known as Chrysler Comprehensive Compensation (or 3C).

Anecdotal evidence from the software industry suggests that XP is a successful methodology. The evidence from various software industry news sites [5] prompted educators to look at XP. Most research has focussed specifically on the paired programming aspect of XP.

At the University of Utah Williams carried out investigations to determine educational benefits of pair programming [13]. She found that two programmers working as a pair were 40% quicker than a single programmer working alone on the same problem. She also found that pair programming improves design quality, reduces defects, enhances technical skills, improves team communications and is considered more enjoyable at statistically significant levels. Nawrocki [43] makes a comparison of XP using pairs, and XP without pairs. In contrast to Williams findings, Nawrocki's experiments showed there was nearly no difference in development time between individual and paired programming. He did find that the pairs had fewer mistakes in their code, and produced more efficient code (measured in lines of code). These improvements were small though. He was unable to replicate the reduction in program development time that Williams reported.

From an educational viewpoint, bug free code, efficiency and short development cycles are not of high importance in first year programming courses. It is important that the students learn to program, that they enjoy programming and that they are prepared for a job in industry. An additional benefit is that there is a reduced workload for teaching staff.

In pair programming partners share knowledge with each other - useful tricks and features of their IDE (Integrated Development Environment), useful programming libraries, good programming practices. Even senior programmers can find themselves learning from more junior developers. Audi [3] wrote "Nearly everyone is sometimes a teacher and sometimes a student". The partners alternate, one moment you are learning from your partner and the next you can find yourself teaching to them. Empirical evidence supports this. For example, McDowell [38] finds that students that learn through pair programming perform comparably in exams to students who learn through individual programming and significantly better when results are adjusted for attrition rates.

The main advantage that McDowell highlighted in "The effects of Pair-Programming on Performance in an Introductory Programming Course" [38] was fewer students dropped the course. He says, "It appears plausible that as a result of pair-programming, students that might otherwise have dropped the course, completed the course." One possible reason is that Pair-Programming causes students to interact more and feel that they belong to a course.

Pair programming can help prepare the students for a job in industry. Software companies are increasingly looking for programmers who can work as part of a team. Pair programming helps to foster these skills in the students.

In addition pair programming can put less pressure on resources. The finding of the experiments in the University of Utah suggest that students do in fact demand less tutor time when pair programming [13]. Also fewer computers are required.

Another positive aspect of pair programming is pair pressure [61]. This pair pressure keeps the students on task and focussed. Students feel a commitment to their partner and are therefore less likely to miss an assignment deadline. This takes a burden off the course administrator who would be responsible for keeping the students focused and holding them to the assignment deadlines.

It has also been observed that pair programming reduces cheating [62]. The pair pressure causes the students to budget their time better. Additionally they have a partner to turn to for help if it is need, and so will be less likely to resort to plagiarism.

Pair Programming and Peer Assessment

Pair programming has a special relationship with peer assessment. With pair programming the feedback from your peer is coming in real time. The partners switch roles from driver to navigator and from teacher to student. Thus, one of the partners is continually assessing the other's work.

So, through pair programming students get feedback on their work from someone other than themselves or their tutor and they get to see the work being done by some other student. These are exactly the same benefits that we listed for peer assessment except that students do not get an insight into the criteria that is used for marking an assignment.

We can consider pair programming to be a form of continuous peer-assessment. The student's mistakes can be caught early, and this helps prevent the formation of bad habits, which can be a continual source of problems during the student's course of study.

Feedback Technology

There are a number of projects using technology to assist tutors in this process of providing feedback to students and in this section we discuss the most relevant projects.

Packages like Markin [14] are available to lecturers or tutors that want to download these tools and install them on their computers. Markin provides tools for annotating and correcting students' assignments. Everything is done electronically. The student can email the lecturer or tutor their assignment. Personal feedback is added to the assignments using the Markin software and finally the feedback can be emailed instantly back into the students inbox. The drawback of these types of software is that the tutor must install special software on his computer. Also the tutor still has to try to organise floppy disks or email attachments with student assignments.

Softboard [30] is a web-based application sharing system that provides a convenient writing and sketching tool in delivering instruction for distance education over the Internet. The system is completely web based and it eliminates the problems of managing floppy disks or email attachments. In experiments the system received very positive feedback from the users. The main concern with this system is that special hardware is required to fully use the system. It uses a special light pen, which is a hand held electro-optical device which when being touched to or aimed at a computer monitor can determine coordinates of the pointed location.

Courseware tools like WebCT [59], Blackboard [10] and Moodle [39] provide feedback features including bulletin boards, collaborative work environments and online quizzes that can help the tutor provide feedback and assessment for their students. They are all very useful, and fully integrated into their web-based environments. However, feedback could be more comprehensive, more student-friendly and more personal.

Studies in the University of Warwick [54] have shown peer assessment to have a positive effect on the students' learning. They developed a novel web based peer marking environment and employed it to help deepen the students understanding of computer programming. 80% of the students that used the system agreed that seeing good and bad programs helped them in learning to program, and that marking helps them to think more deeply about their work.

Further work [8] in the same University found that peer assessment could be used to successfully reduce the resource requirements for administering and marking laboratory tests. It was also beneficial to the learners in improving their critical and analytical abilities in programming and problem solving.

The University of Glamorgan [34] has introduced an "add-on" to the Coursemarker Programming Environment that permits the use of student peer-assessment. The students are encouraged to use the system by gaining extra credit for providing good feedback. An automatic program grades their feedback to determine this extra credit.

Conclusion

Feedback, peer-assessment, self-assessment, pair programming, and individual instruction will help student's motivation and deep learning. We believe that this is why our system, which relies on feedback in all the forms discussed in this chapter, can help students to learn. Other feedback systems have been useful but have failed to provide students with the level of feedback necessary. Individual feedback in a timely manner is possible to achieve even for large classes and this thesis evaluates our system that does just that.

Chapter 3: Methodology

Introduction

The main focus of this research was to evaluate the use of an administration system to manage individual feedback during a programming course. In order to do this a prototype was developed, which we called TutorBoard.

Throughout the thesis 'we' will be used to refer to my supervisor and myself.

The methodology that we used to evaluate our system was a positivist approach. For our control group we chose to use the previous year's students. This group consisted of 277 Programming students. The experimental groups had 124 students for 2002 and 210 students in 2003. For each of the three years the course material was the same, the same notes, the same lecture/lab format and similar lab exams and end of year exams. As the class sizes are relatively small the results were rather limited but we did find a statistically significant positive relationship between the users of the TutorBoard and the end of year results.

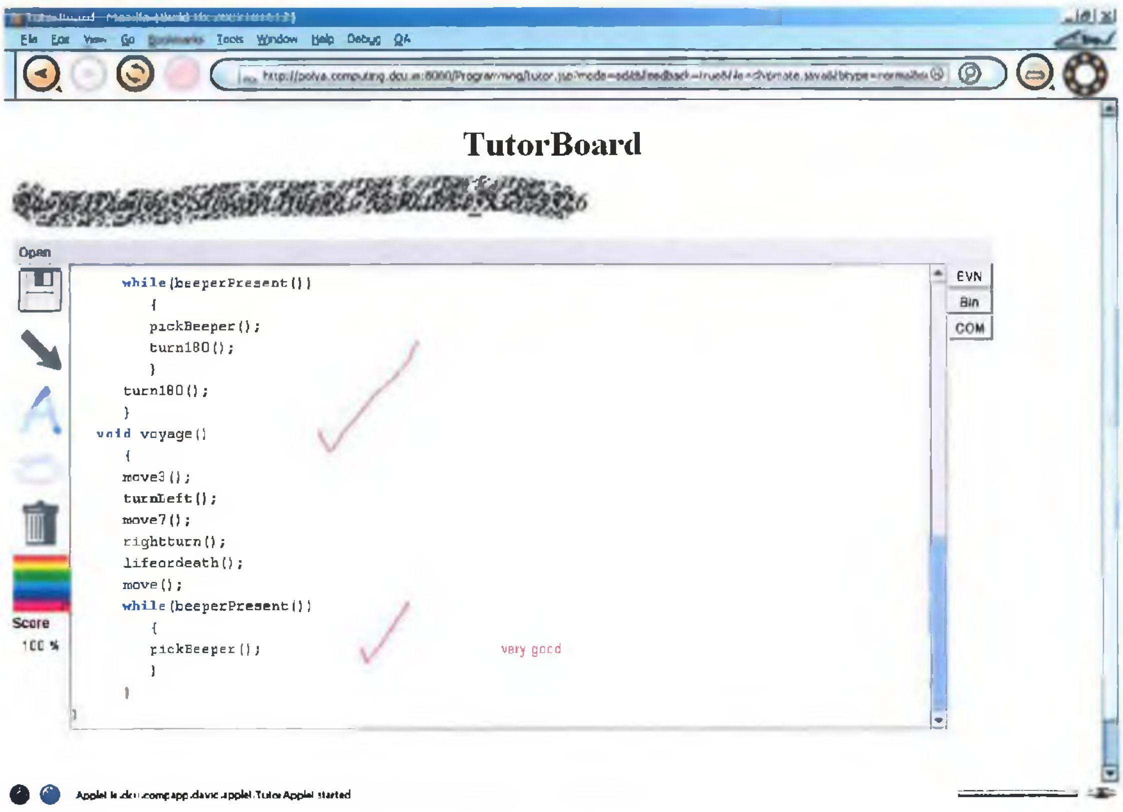
As well as this positivist approach we collected opinions using questionnaires to help understand the results.

The Tools

TutorBoard was designed to be completely web-based and therefore accessible from any computer. It encompasses all the areas discussed in the literature review, comprehensive tutor feedback, pair programming, peer-assessment and self-assessment.

We wanted the system to be easy to use. There have been a number of guidelines produced for good interface design. For example IBM (1997) [20] recommend paying particular attention to the following principles: simplicity, support, familiarity, obviousness, encouragement, satisfaction, accessibility, safety, versatility, personalisation, affinity.

The Tutorboard users were all computer science students. This made it easier to design the web interface, as these students would be already very comfortable with using the web. The design concentrated on making information easy to find with as few clicks as possible. There was an emphasis on familiarity, in particular, for the feedback. Students are used to receiving feedback in the form of red marks and comments written over their own work. This is the format we wanted to mimic (see Figure 2).



[redacted]

```
      while(beeperPresent())
          {
          pickBeeper();
          turn180();
          }
      turn180();
      }
   void voyage()
      {
      move3();
      turnLeft();
      move7();
      rightturn();
      lifeordeath();
      move();
      while(beeperPresent())
          {
          pickBeeper();
          }
      }
}
```

very good

Applet ie.dcu.compapp.davic.applet.TutorApplet started

Figure 2: Sample of familiar feedback (feedback produced by a tutor in 2003 for an assignment on programming a robot to move around a map)

TutorBoard took a lot of the normal administration tasks away from the tutors and the lecturers by dividing the class into tutor groups. All that the lecturer had to worry about was monitoring the tutors, while all that the tutors had to worry about was monitoring their own group of students. It used peer assessment to set up a hierarchy that was easy to manage using divide and conquer.

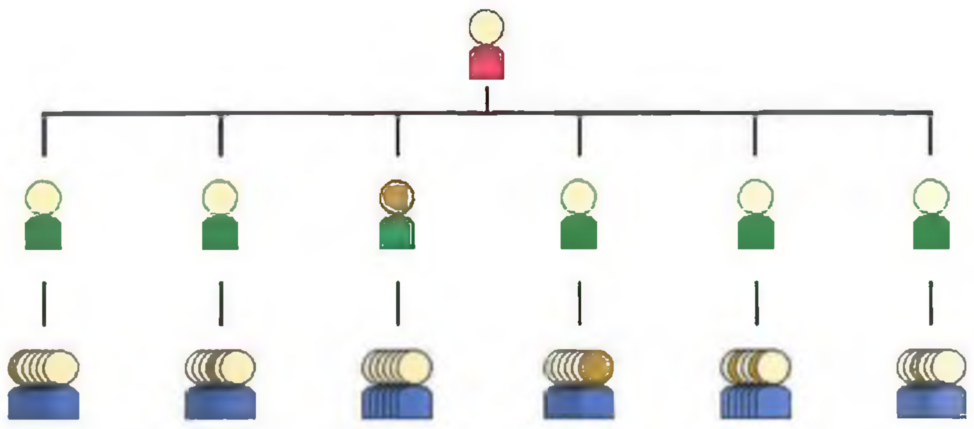


Figure 3: Divide and conquer to manage large classes – a 3-tier representation of the class, the lecturer is on top, he is in charge of a group of tutors, and finally on the bottom tier each tutor has a small group of students.

Students

TutorBoard enabled the students to:

- Read the assignment specification.
- Upload their submissions.
- Review all the files that they had submitted.
- Quickly access their feedback from their tutor.
- Provide feedback on their peers.
- Provide feedback on their own assignments.
- Access the feedback provided by their peers and themselves.

The TutorBoard system encouraged the students to provide feedback. It hid the tutor's feedback and the tutor's mark, and prompted the students to complete the assessment process, shown in figure 4. By completing the simple task of assessing one of their peers and themselves the students were rewarded by getting their feedback.

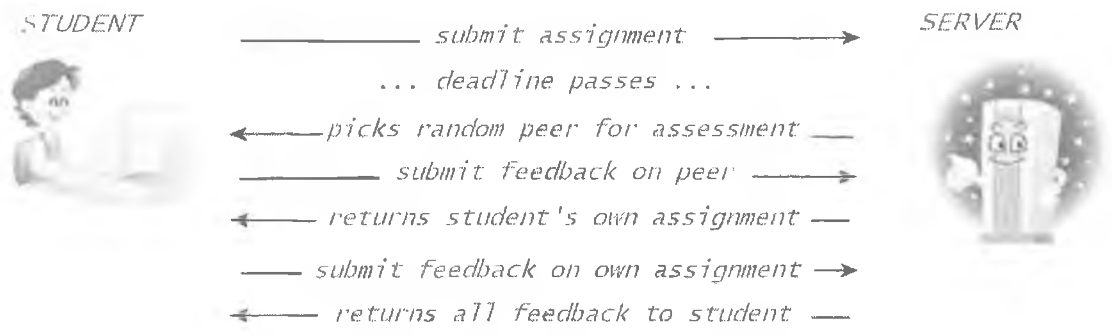

STUDENT
SERVER
submit assignment
... deadline passes ...
picks random peer for assessment
submit feedback on peer
returns student's own assignment
submit feedback on own assignment
returns all feedback to student

Figure 4: Student/Server interaction

Furthermore, the students who gave marks to the assignments that closely matched the official mark given by the tutor to that assignment received bonus marks. The maximum bonus they could get was 10. The bonus marks were added on to their assignment score. This discouraged them from just providing meaningless feedback.

So, as discussed above, we provided two clear incentives, which cater for the diversity of the students in the class. Extra credit will encourage some students, but receiving feedback will motivate others.

Tutors

The tutors were chosen from the top of the class of the previous year. This is a completely scalable solution to managing large class sizes; no matter how large the class if you take the top ten percent students of the previous year as tutors you will have a one to ten tutor to student ratio. This is only true if the class does not grow significantly each year. Also the second year tutors are familiar with the course, the syllabus, and the common pitfalls. This meant they could give more satisfactory feedback.

In addition, getting feedback from second year students is good because the first year students view them as a role model. This helps in two ways. Firstly, students know that there is a tangible reward for being in the top 10% of the class – they may be tutors in the following year. Secondly, they realise that they could be as capable as the tutors are if they work seriously at the course. This means they will value the course and expect to succeed – these are the two factors that drive motivation [25].

Tutors could manage their group of students completely over the web-based system. They could review the students' submissions, run their programs (figure 5) and mark them (figure 6) online. Any sample answers and marking schemes were made available to the tutors by the system.

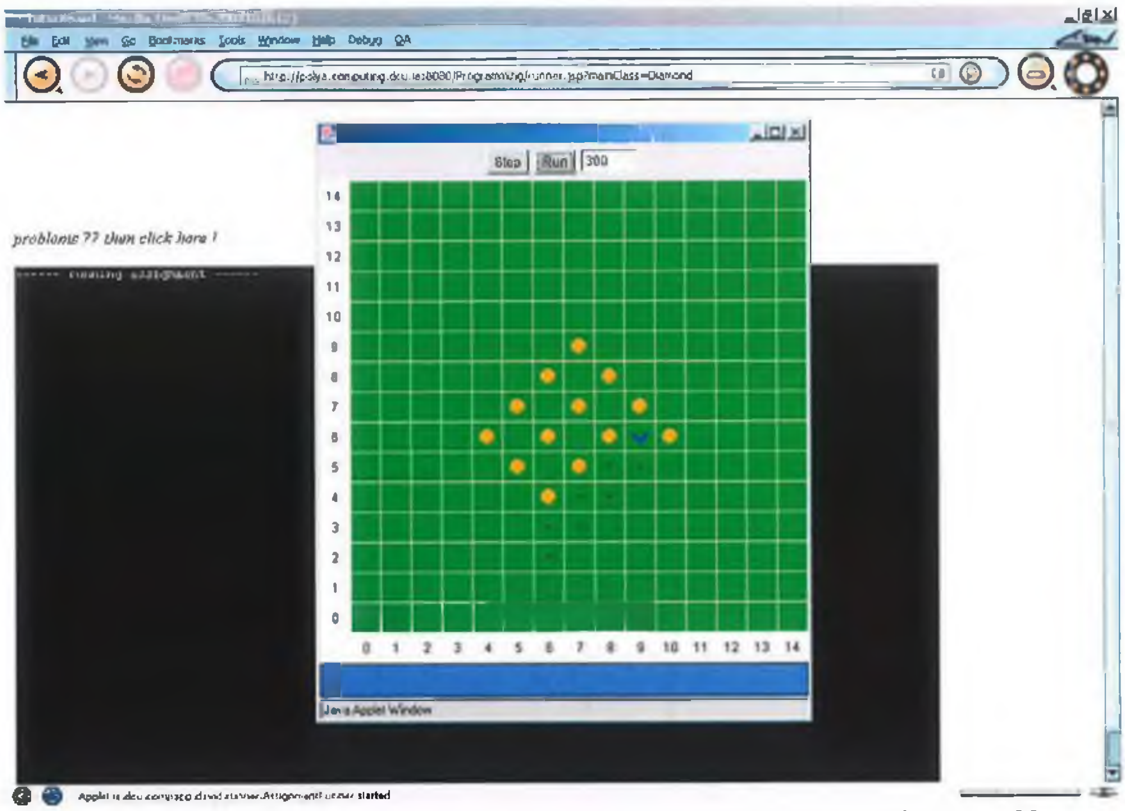
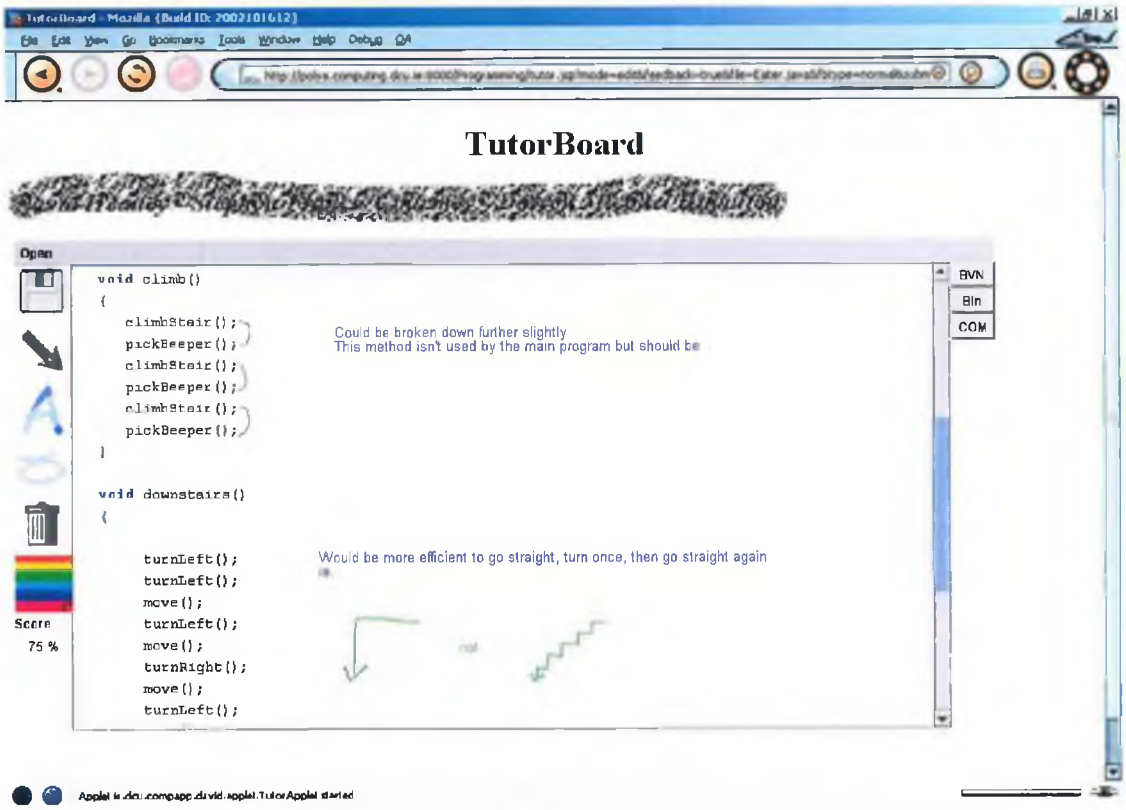

File Edit View Go Bookmarks Tools Window Help Debug QA
http://polya.computing.dcu.ie:8080/Programming/runner.jsp?mainClass=Diamond
problems ?? then click here !
------ running assignment ------
Step
Run
300
Java Applet Window
Applet ie.dcu.compapp.david.runner.AssignmentRunner started

Figure 5: Tutor running an assignment (the assignment was to write a 'robot' that would pick up the specific arrangement of 'beepers')


TutorBoard - Mozilla {Build ID: 2002101612}
File Edit View Go Bookmarks Tools Window Help Debug QA
TutorBoard
Open
BVN
Bln
COM
Score
75 %
Applet ie.dcu.compapp.david.applet.TutorApplet started

[redacted]

```
void climb()
{
    climbStair();
    pickBeeper();
    climbStair();
    pickBeeper();
    climbStair();
    pickBeeper();
}

void downstairs()
{

      turnLeft();
      turnLeft();
      move();
      turnLeft();
      move();
      turnRight();
      move();
      turnLeft();
```

Could be broken down further slightly
This method isn't used by the main program but should be

Would be more efficient to go straight, turn once, then go straight again

Figure 6: the feedback that the tutor gave on the student's code for the assignment shown running in Figure 5.

Lecturer

The lecturer used the system to monitor the tutors. He could read the feedback they had given to their students. He could also provide feedback on the feedback that the tutors provided. He had access to all the students' grades and bonus marks. He could manage everything assignment related by supplying a specification file, the deadline, a sample answer, a marking scheme, whether the assignment should be done in pairs or individually, whether it was a self-assess assignment and whether it was a peer-assess assignment. Also the lecturer could provide extra feedback and modify marks if required.

There was no need for the lecturer or course co-ordinator to manage the students' passwords, or user accounts. All this was managed by the system. The system could query the University's LDAP server (or password server) to verify the user's passwords. This also made things easier for the students, who had a common password for all the resources they used in the University, including our TutorBoard system.

The Students

The main people involved in the evaluation of the system were the students and tutors. They were asked to use the system for their lab work. The trial was separated into two different trials over two years. The students in the first year evaluated the system based solely on personal and timely feedback. While in the second year the students/tutors evaluated the peer/self assessment aspect of the system. This approach was decided upon after the success of the first year's trial. Feedback from the students and tutors prompted us to add extra features to provide more feedback. And so peer/self assessment was added and evaluated for the second year.

Collection Methods

We used four sources of data in evaluating the system.

1) Log files.

The system produced basic log files that recorded the students' usage of the system. This data could be analysed to measure the student's enthusiasm and willingness to use the system. The system recorded when a student logged on, when a student submitted a file and when a student accessed feedback. As these logs were produced automatically by the system it was possible to ensure that they were a true reflection of the use of the system.

2) Exam Results

We were able to compare student performance in the assignments with their end of Semester exams. Also for the first trial we had access to the Leaving Cert results. This gave us the opportunity to validly compare the marks for students with different academic abilities.

3) Questionnaires

The questionnaires were handed out to the students and tutors. To ensure that we got a representative response we handed these out during the lab sessions. Attendance in the labs can drop off for the last couple of lab sessions so we circulated the questionnaires early enough while the attendance was still high, and late enough so that the students had had plenty of time to get used to the system.

The responses, which comprised quantitative data (Multiple-choice questions) and qualitative data (comments/opinions), were all processed. Some of the comments are used to illustrate the student's perspective.

The sample questionnaires are in Appendix C.

4) The feedback

All the feedback was saved on the server and it was possible to analyse this feedback. In order to analyse the feedback extra software was developed that would read the feedback and classify the type of feedback that each tutor gave, and that each student received.

(Sample feedback is in appendix D.)

Further details on data sources and analysis will be discussed in Chapters 6 and 7.

Conclusion

The positivist approach gave this research validity and rigour. The control group we used was the class of 2001. They studied the same course. The Tutorboard was the key difference in teaching methods between the control group and the experimental groups.

The prototype system that we developed fully supported the teaching practices discussed in the literature review, i.e. it supported timely, comprehensive and individual feedback to a large class of students and allowed teaching by using Pair Programming. The system was improved by adding peer and self assessment for the second trial. The first trial focussed on evaluating individual feedback. The second trial focussed on evaluating the new feature – peer and self assessment.

To fully evaluate the system we analysed log files, exam results, questionnaires and the feedback itself. The results will be discussed in the Chapter 6 and Chapter 7.

Chapter 4: Technologies

Introduction

The TutorBoard system described in the previous Chapter was built upon free and open software components. In this chapter we discuss all the major technologies that we used. The client side and the server side were developed using Java. The annotation whiteboard on the client's browser used Java's Applet technology. The server side which managed the students and all the feedback files used Java's Servlet technology. The Servlet ran on a Tomcat Servlet engine and connected to a MySQL database which contained the application data. We chose to run the server on both the Windows operating system and the Linux operating system.

Java

Java was developed by Sun Microsystems in the early 90's and has now evolved into a robust and versatile programming language. With Java, developers can write software on one platform and run it on another, they can create programs to run within a web browser and they can develop server-side applications. Sun now offer a variety of products built on their Java programming language. These include the Java Standard Edition, the Java Enterprise Edition, the Java Micro Edition and Java card technology. The Standard Edition includes Java Web services technology like Applets that met the requirements for this feedback system project. The Enterprise Edition includes enterprise applications such as Servlets, which were also useful for the type of project we were undertaking. These are the two main Java technologies we used.

Applets

Java ‘Applets’ are mini applications that your web browser downloads and are run within the browser. The advantage of writing our feedback application as an applet is that applets can be run on any computer (with a web browser) without having to download any special software. Applets make your system usable on any computer, on all platforms. So, Java enabled the tutors to annotate the students’ work from anywhere.

One problem with applets, though, is that different browsers support different versions of Java, see [45] and [52]. Our TutorBoard system requires the users to have the Java 1.4 plug-in installed on their machines and to configure their browser to use Sun’s jdk to run the applets.

Servlets

Java ‘Servlets’ are similar to applets but instead of running on the client’s machine (in their browsers) they are run on the server machine. They make developing complex server code very straightforward. Our Servlets were able to manage all the requests to the web application – managing login sessions, uploading assignments, uploading feedback, reviewing feedback etc.

JSP’s are a newer Java technology that builds on the power of Java Servlets. JSP stands for Java Server Page and is similar to Microsoft’s Active Server Page (or ASP) technology. ASP is only supported by Microsoft’s IIS web servers. However, nearly every popular web server can be configured to run JSP, including Microsoft’s IIS, Netscape Enterprise web servers and Apache web servers. This is a clear advantage as Microsoft has about a 20% share in the web server market, where Apache's share is approaching 70% [44].

We can sum up what JSP's are in one sentence. They are Servlets that are compiled by the Servlet Container automatically when the source is updated. In this way they are more suited to dynamic content. We used JSP pages to present the information and manage the user interface. As Sun's J2EE Blueprints [22] recommends, we used Servlets strictly as a web server extension technology. This included the implementation of specialized controller components offering services like authentication, database validation, and so forth.

There are other technologies that offer a lot of server side power such as php and any sort of cgi scripting. Technologies like Javascript also provide some client side processing power. However, Java had other benefits, e.g. it is an object oriented programming language. Object oriented languages define 'Objects' that are used to manipulate and process data. The more traditional way was to view a program as a logical procedure that would take data, process it and produce output. The advantage of object oriented programming is that it is easier to re-use code. 'Objects' can be extended and modified easily without having to re-write reams of code. This means that there already exists lots of code that is easily modifiable and completely re-usable that will already do some of what we want our system to do.

Basically Java was the all round package. It has useful server side technology, useful client side technology and it is a popular language meaning that lots of libraries and re-usable code already existed. Being able to write the entire project in Java meant that the client side and server side could seamlessly fit in together.

Tomcat

Since we decided to deliver our web application as a system of Servlets, we needed a Serlvet Container to run it in. Popular Servlet Containers include ColdFusion, Tomcat and Resin. Tomcat [2] is the servlet container that is used in the official Reference Implementation for the Java Serlvet and Java Server Pages technologies. Tomcat is developed as part of Apache's Jakarta project.

We decided to run Tomcat on our server machine. All our web applications were run on it and it served all our web pages. There were many advantages in using Tomcat over other servlet containers. Firstly Tomcat is the official reference implementation for Java Servlets, which guarantees that the Java web applications will behave as expected. It is developed by some of the top developers in the world and is used extensively over the Internet. For these two reasons it has evolved into a robust and reliable piece of software. In addition, it can run on any platform, since it is written in a platform independent language – Java. Finally, unlike the ColdFusion servlet container, Tomcat is free to download and use.

MySql

We needed a suitable database server that would manage our data for the web application. We did not have major requirements for the database. The database was needed to save the user's profiles, for example, the student's tutor, the peers they had to mark, whether they submitted an assignment, whether they had marked an assignment etc.

According to their website, MySql [42] is the world's most popular open source database. It is free to use and is available for the main platforms, including Linux and Windows. Furthermore you can connect to a MySql database server from all of the major platforms, using nearly any programming language.

It more than satisfied our requirements.

Linux/Windows

It was important to find a secure and reliable platform to run all these on. The first choice was Microsoft Windows. Windows ([7] and [63]) is one of the most popular operating systems. It was first released in 1985. The initial version of Windows was an extension of the DOS operating system that provided a graphical operating system for PC users. Since then it has undergone many improvements and is now the most popular Operating System.

Windows popularity means that crackers can create a lot of havoc by targeting this one Operating System. Crackers have developed numerous viruses and worms that affect Windows, making it possibly one of the least secure operating systems. With over 200 computing students using the machine, security is an issue. If they compromised the system they could possibly change their grades or plagiarise other student's assignments. This was an especially pertinent problem since we wanted the machine to be accessible from outside the University, to accommodate students who wanted to work from home.

Linux [35] was another option. Linux was initially created as a hobby by a young student, Linus Torvalds, at the University of Helsinki in Finland. Linus had an interest in Minix, a small UNIX system, and decided to develop a system that exceeded the Minix standards. He began his work in 1991 and the current full-featured version was released in January 2001. Development continues with the assistance of top developers all across the world. As a free alternative to more expensive operating systems it has become very popular of late. For us it would seem to solve our concerns over security.

In the first semester-long trial we used Windows. We had very few problems. There was one case where the server was compromised but no lasting damage was done. The server was not just running the feedback system. The server was running other web applications in addition to the feedback system, e.g. a bulletin board. Tomcat running on Windows seemed to be able to handle the load fine. At peak times, the TutorBoard feedback system registered over 100 login sessions per hour.

For the second semester-long trial we moved it to Linux. One problem that we encountered with Linux was due to Linux's tight security rules. Linux will not allow root access to its graphic's server, and neither will it allow remote programs access to it. The solution to this problem was neat and illustrates nicely the flexibility of Java, which was one of the main reasons we developed the system through Java. The interested reader can see an outline of the solution in Appendix A.

This platform independence that Java gives to the system offers the course coordinator the liberty of selecting their preferred operating system. On the client end the system was also totally platform independent. Any browser supporting the latest version of Java is all that was required. This enabled the tutors to mark the assignments equally with Opera, Mozilla, Internet Explorer or any other popular browser on any operating system. The students could access the feedback just as easily.

Conclusion

The TutorBoard feedback system was built on all these technologies. Our knowledge of these technologies and how best to use them helped of course, but without the underlying technology it would not have been possible. There were many technical problems encountered along the way. Technically minded readers will find the details in appendix B.

Chapter 5: Lab Tutor System

Introduction

Before we see how the TutorBoard feedback system impacted on Programming courses here in DCU, we will discuss the systems already in place. The lab tutoring system allowed 2nd year tutors to supervise small groups of 1st year students and give feedback to the lecturer on how each student was progressing. The feedback was given by using a web based feedback system.

This lab tutor system remained in use for the TutorBoard trials, thus maintaining the validity of the study. The key difference between the control group and the experimental groups was the use of the TutorBoard.

Lab Tutoring System

A novel tutoring system was in use in DCU to combat the problem of managing computer lab session for large class sizes. This problem was targeted because it was believed that poor lab management contributed to the low pass rates and the poor programming ability displayed by many students after taking a computing course. The student must learn problem solving and abstract reasoning in order to become a proficient programmer. The lab environment is an appropriate place to learn these concepts.

A significant drop in entry points for the course made the problem of managing the labs more important. These students with low points have been singled out as the ones most at risk of failing the course [40].

Firstly, the lab sessions were completely restructured. The timetables were changed so that the labs followed the lectures and all lab sessions were held simultaneously. In addition, second year students were employed to tutor the first years. Each second year student tutored a bay of ten students. They also attended a training course for tutors prior to the start of the semester. Finally a web based feedback form was used to monitor the tutors, and lab sessions [15].

The Tutors

The tutors were taken from students of the previous year's class who had the top marks. These second year students knew the material and because they had covered it recently they would be able to remember the problems that new students have with learning a programming language. Each tutor managed a group of about 10 students. This ensured that tutors were able to give individual attention to the students.

This involves dedication from the second year tutors, who are busy with their own modules. However, the time spent on this extra work is worth it because they learn relevant skills from tutoring. Teaching to their students reinforces the ideas in the tutor's own mind.

One important issue is the cost. All the tutors were paid the standard rate for their work. However it ensured that the students received individual attention in the lab. Another advantage is that the extra money the tutors receive for doing extra course related work will help them out. Financial problems are often the cause of students failing to complete a course of study. However if cost to the institution is a problem it still might be worth considering giving academic credit to the tutors rather than direct payment. Students in Stanford have participated in such programs where they receive credit for taking part in a tutoring or mentoring program [50].

Training

The second year tutors took part in a short training course. During the course, specific tutor qualities were emphasised. These qualities were patience, friendliness and approachability. In addition the tutors were taught that they must help the student to solve the problem, rather than actually solving it for them.

The training course also included role-playing sessions. The students were divided into groups of three. One student would play the first year student, one would play the tutor and the other would be the observer. They were given situations to play out. For example:

Student: Yesterday you missed the lab ‘cos you have a job. But if you’re not signed in for the lab, you could get in trouble with the college and with your parents. Luckily you see the lab tutor in the corridor and approach him to get him to sign you in.

Tutor: A student, missed a lab and wants to talk to you about it.

Observer: Ideally the tutor needs to be patient but firm. Also the emphasis should be on explaining the importance of the labs rather than the impossibility of signing him in.

Figure 7: Tutor training using role-play scenarios.

Web based Feedback Form

The tutors reported back any problems with the labs or any problems specific to any student back via a web based feedback form. This system enabled the tutors to keep an attendance of the students, give specific comments on the progress of each student and give comments on the lab session as a whole. The lecturers and course co-ordinators had access to all this information.

With this information the lecturer can take positive action if particular students seemed to be having trouble. If specific problems are repeatedly reported in the lab then the lecturer can adjust the pace of the lecture or go over a certain aspect of the course again. The comments on the labs help improve the labs for the following year. The comments about individual students may help to identify potential tutors for the following year also.

Results

There was a positive response to the system from both the tutors and the students. The students appreciated the extra tuition and the non-intimidating atmosphere in the labs.

Typical comments were:

"I enjoyed the friendly relaxed environment."

"Help is at hand when you need it."

For the tutors the benefits were financial, academic and social. They met more tutors, there was no need to get a part time job and they got plenty of revision of their java programming.

Representative comments from the tutors were:

"I enjoyed getting to know more people."

"It gave me a better understanding of last year's work."

"Getting to go over the first year course has really improved my knowledge of Java."

"Satisfaction of helping students understand."

"Not having to get a part time job."

For the lecturer or course co-ordinator the benefits were that they were more in tune with what was happening in the labs. They could schedule special tutorials for the struggling students and/or could change the lecture material to more suit the progress of the labs, as already mentioned.

Finally, all these benefits manifest themselves in terms of improved results. The failure rates dropped when the system was employed. In addition, the encouragement of social interaction in the labs appeared to be of benefit, in particular to the female students. This social interaction improved the lab environment.

We were able to manage large classes and increase the number of tutor hours each student received. Naturally, this was a very successful system. However there was one part of the course we were not happy with – assignment feedback.

Conclusion

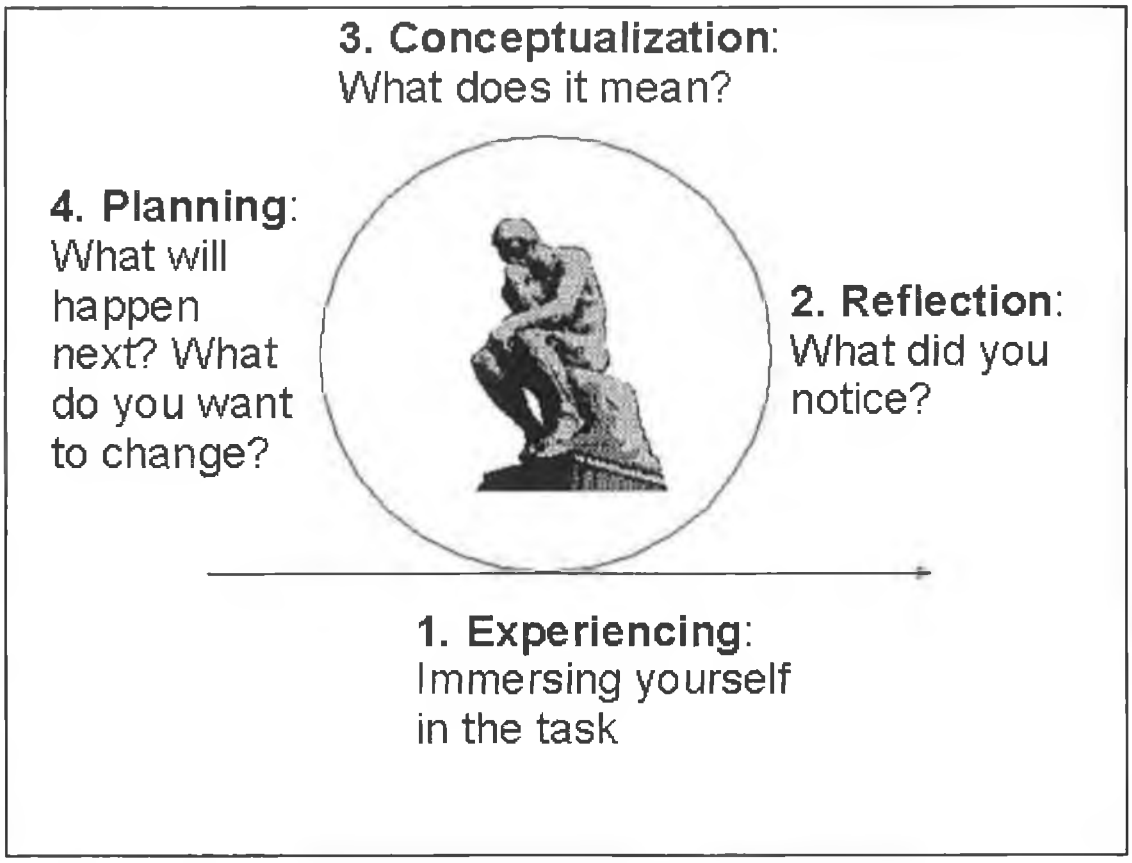
The system described in this chapter did make a large class more manageable by breaking it into smaller groups. The students were more open to advice from their near peers (the 2nd year tutors). The tutors benefited from the extra money and from revising the material by going through it with the students. The atmosphere in the lab was pleasant for all involved and appropriate for learning.

There remained a problem with giving feedback to the students on their assignments. The TutorBoard system would solve this problem. The TutorBoard was used alongside this system. This meant that the one single difference in the teaching methods between the control and the experiment was the use of the TutoBoard. This makes our comparisons between the control group and the experiment group valid.

Chapter 6: Mass Production of Individual Feedback

Introduction

The importance of feedback has already been discussed. However, we now want to look at it in more detail. We can consider Kolb's 4 stages of learning [29], depicted in the figure below. I am using Kolb's cycle below to illustrate how a student uses feedback (which is the main focus of the study) to learn.



3. Conceptualization: What does it mean?
4. Planning: What will happen next? What do you want to change?
2. Reflection: What did you notice?
1. Experiencing: Immersing yourself in the task

Figure 8: Graphical representation of Kolb's learning cycle (the famous Thinker Statue by Auguste Rodin is depicted in the centre of the cycle).

Without feedback the students' learning process is cut off after stage 1. In other words they do their assignment, submit it and then forget about it. Clearly this is not how to learn and students will quickly lose interest in doing the assignments that the lecturer sets them.

The web based TutorBoard (described in Chapters 3 and 4) feedback system was developed to address this problem. It built on the lab tutoring system of Chapter 5. We continued to use the second year tutors and we continued to use the web feedback form. However, added to this we deployed an online system that the tutors could use to provide feedback to the students on their weekly lab assignments. This Chapter will discuss how well the system worked.

Autumn 2002

On Thursday, October the 10th 2002 the first official user, a first year Computer Applications student called Lorcan, logged on to the TutorBoard system at 2 minutes after 11 to check his assignment specification for the labs for that week.

This was the beginning of the TutorBoard system's first trial run in Semester 1 in the Introductory Programming course run by the school of Computer Applications in DCU for first year students. The majority (138) were studying a Computing degree but there were also some Mathematics (36) and Computational Linguistics (16) students. In total there were 209 students taking this course. The course taught programming through Java. The assignments generally required the student to write full Java programs.

The students were split into groups of about 9 or 10 for the lab sessions. Each lab group was assigned the same tutor for the entire semester. There were 23 tutors in all. The tutors were given a short induction course to introduce them to what was required of them (see the previous chapter for more details of the induction course).

Over the 12-week semester there were 2 two-hour lab sessions scheduled each week. One was on Tuesday afternoon (2pm-4pm) and the other was on Thursday morning (11am-1pm). Average attendance at the labs was 85%.

On Tuesday the students were given an assignment to complete on their own. On Thursday they were paired up and given an assignment to do together. This paired assignment was corrected on the TutorBoard. Revision was scheduled for some weeks instead of this paired assignment. They could generally choose their own partner and it need not be the same partner from week to week. During the semester, they had 7 paired assignments to do. The tutors instructed them on how to pair program effectively. The students worked together on the assignment during the lab session. The assignment should then be submitted through the TutorBoard before the next Tuesday, and then the tutors could mark it and by the following Thursday's lab the students would have their feedback.

Extensive logs were kept of how the system was used, for the purpose of evaluating the affect the system had on learning. Also during the second last Thursday of the semester a questionnaire was handed out to the tutors. Every tutor returned the questionnaire. On the final week a separate questionnaire was handed out to the students who used the system and were present in the lab that week. We received 128 responses, which represents 61% of the students registered on the system.

Analysis of Usage

The extensive usage of the system proves that the students valued receiving their feedback. In figure 9 you can see the number of logins by day (totalled over the 12-week Semester). Tuesdays and Thursdays were the busiest days as those were the days the labs were scheduled. The graph does show that the system remained busy throughout the week and was also used during the weekend.

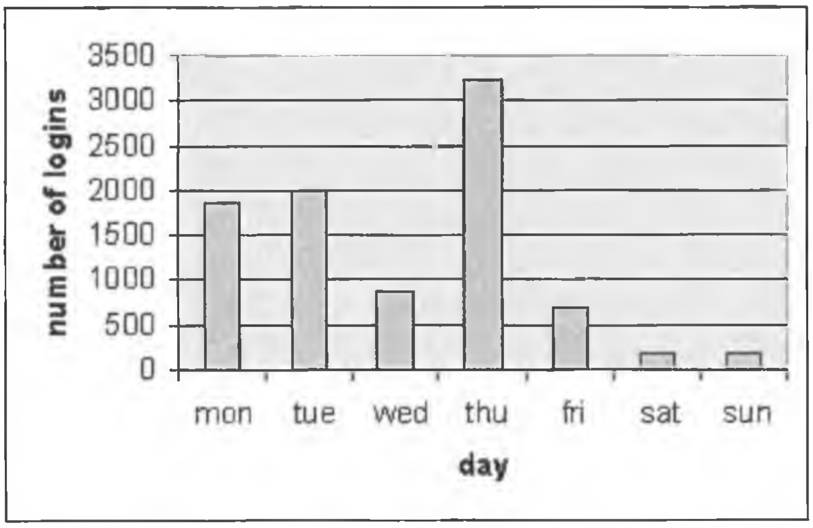


Figure 9: Total number of logins over the semester (by day) 2002

Looking at the time of the day the students logged in shows a similar trend. Obviously during the lab sessions, people were encouraged to log on and they were the busiest times i.e. from 11am to 1pm and from 2pm to 4pm. However the system still remained busy after 4 o'clock in the afternoon right up until about 8pm. There were also some sessions at 1am and 2am in the middle of the night.

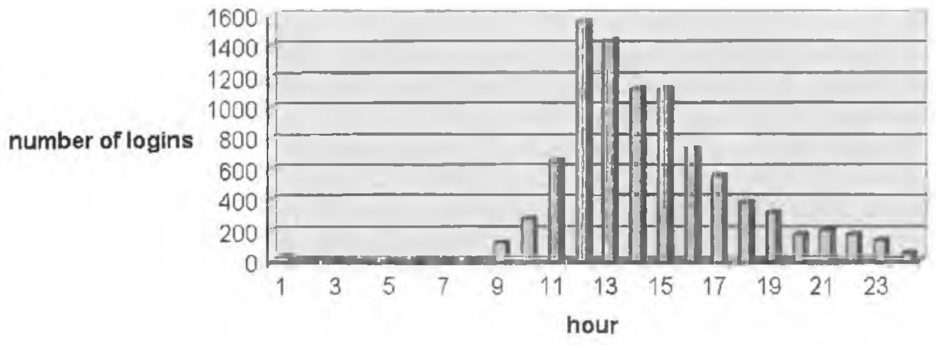


Figure 10: Total number of logins over the semester (by time of day) 2002

Logon sessions in the evenings and during weekends proved that students were getting quicker access to their feedback than they would be through traditional assignment submission processes.

Figure 11 shows how the users became more comfortable using the TutorBoard system. The students were introduced to the TutorBoard on week 2 of the semester. Week 3 was the busiest week with over 1500 login sessions. By week 4 the students were comfortable with the system, and the usage statistics were consistent from then on. There were about 600 login sessions per week. The last week of semester predictably the number of logins falls off, as students are starting to revise for exams and there was no assignment that week.

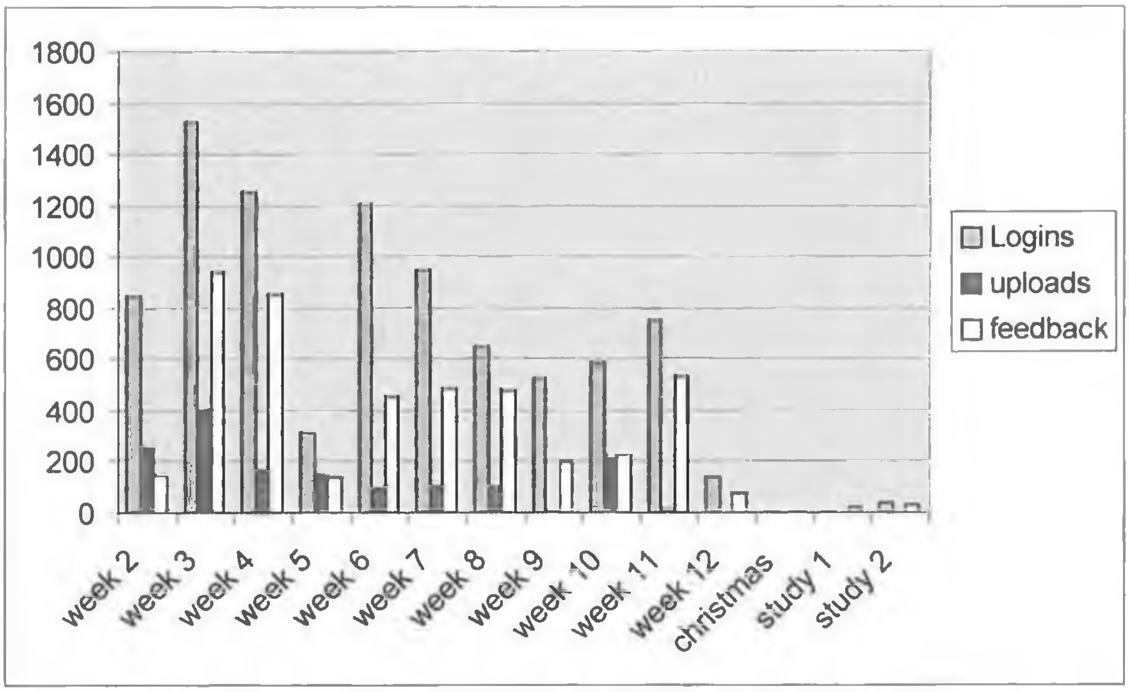


Figure 11: TutorBoard Usage over Semester 2002

The graph also investigates what the TutorBoard was used for. The uploads indicate the number of requests to TutorBoard to upload a file (part of an assignment). Some weeks there were no uploads because some weeks there were just no assignments. On an average week there were about 100 file uploads, which makes sense since there were roughly 200 students paired up for each assignment.

The feedback illustrates the number of requests to TutorBoard for feedback files. The TutorBoard would get a request for feedback when a tutor marked an assignment or when a student viewed the feedback for his assignment. The graph indicates that students regularly accessed their feedback.

The tutors found the system useful for marking the student's assignments. They made good use of most of its features.

The following table presents a global overview.

Java sources submitted	992
Java sources marked	842
Assignments graded	789
Feedback Components used by tutors	
Text elements	2000
Scribble elements	1589
Predefined comment elements	224

Table 1: Synopsis of tutor feedback

The grade provided was for use only for assessment purposes. But as you can see nearly all the assignments that were submitted were graded (note that the 992 sources submitted does not represent 992 assignments, as some assignments may require two or more source files). Not all tutors provided feedback; in 15% of cases no feedback was supplied. This is an issue that will be addressed in the next version of TutorBoard.

The tutors also varied in the type of feedback that they provided. Few took advantage of the predefined comment elements. These predefined comments would be used more if we added more for some of the more common mistakes that the students made. However, tutors used the text elements and the freehand 'scribble' elements frequently.

Later in this Chapter we discuss the impact of the different types of feedback. For now we will just say that the tutors gave sufficient feedback and the students logged on often enough to read it. The students could access their feedback when they wanted. Many chose to log in at the weekend or in the evening to check their feedback. This is a big improvement on the previous year when the feedback was poor (often merely a mark) and late, typically weeks after the assignment was submitted.

We used the questionnaires to discover the users opinion of the system two separate questionnaires were drawn up, one designed for the tutors that used the system, the other for the students.

Students Opinion

We wanted the system to provide feedback that was timely and clear and relevant to the students' assignments. For the most part the students agreed that we met these targets. 53 out of the 128 responses strongly agreed that the feedback was clear, 36 strongly agreed that it was relevant and 40 strongly agreed that it was quick. In addition, 75 students indicated that they strongly agreed that it was useful to be able to submit code from anywhere at any time.

Although we were using second year tutors, it appeared that they were able to provide helpful feedback to the students. In the survey we had a positive response from the students about the tutors' feedback. Indeed 88% of students agreed or strongly agreed that "the tutors were able to explain the material well" and 97% of the students surveyed agreed or strongly agreed that "the tutors knew the material very well".

They were also asked questions on their experience with the course. Here is a sample of the responses we got when we asked for their version of the best feature about the feedback applet:

"It gave us a good idea about where we're going wrong in our programming"

"It is a clear and precise way of seeing errors."

"it made program very easy to read (with colours etc)"

"I could submit the code from anywhere at any time."

"Easy to use - I was able to check my tutors feedback anytime anywhere."

"easy to use; easily accessed; quick correction of work"

"easy to use, a lot handier than having to go find your tutor"

"It was good to be able to see the advice directly linked to certain parts of the code."

"Getting a pat on the back. We all like the praise we can get, ya know."

"It was easy to use and feedback was returned quickly"

"You could view your programs as you wrote it, with comments pointing to where you could have improved the program."

"You could ask experts about problems ex people in the class who had a higher skill of programming questions, and the answers were good."

It is evident that they liked the simplicity of the TutorBoard feedback system and that it was easy to use. They commented on the convenience of being able to submit assignments from anywhere and check their feedback from anywhere too. They also found the feedback encouraging. On the negative side, there were some interesting replies when we asked for “the worst thing about the feedback applet”. Here is what they said:

“undecided, maybe could have had audio, sound”

“We often didnt get feedback”

“it takes a while to get the feedback”

“It didnt work sometimes.”

“took long to open”

“cant access from home”

“not enough info if something was incorrect”

“Nada, seems pretty cool to me.”

There had been some bugs and problems with the system that surfaced slowly as the semester went on. So there were some students who pointed these bugs out to us here. In addition some tutors were not as good at giving feedback to their students. Some students rarely received any feedback at all from the system unfortunately.

We also asked for ways they thought might improve the feedback applet. They said:

“There was maybe correct copies of code to actually compare yourself.”

“Instant Messaging”

“scores were given on your assignment”

"Tutors themselves submitted a program to each student showing the best and most effective way to do each assignment after you had submitted your program with all the extra credits shown so that students can learn from past mistakes."

"People used it more and were more willing to give feedback"

"it worked more often"

Many of these issues were addressed for the following year.

Tutors Opinion

The tutors filled in a questionnaire on the last week, all tutors were present and all of them returned a questionnaire.

The tutors' responses were also positive. They liked using the system and found it useful for correcting all kinds of errors that the students made. In addition, all but one agreed that they found it useful to provide the feedback whenever they wished.

They were a little more divided but still very positive on the issue of whether they learnt anything from the experience. 14 out of the 23 did say that their Java improved as a result of tutoring and 10 said that their problem solving skills improved. They seemed surer that the experience jogged their memory with 18 agreeing that teaching reminded them of concepts and ideas from the previous year that they had forgotten about.

We also asked some opinion questions of the tutors. Here are some typical replies we got from them on how they felt the system worked.

What the best thing was:

"was able to mark assignments very quickly"

"Easy to use. Save a time."

"You could go through their code in your own time and provide good solutions to their problems. Instead of trying to figure out their code in front of them."

"It was easy to write over the code and show exactly where errors occurred."

What the worst thing was:

"If you didn't understand a piece of code the student wasn't there to explain it to you."

"Sometimes it is not easy and takes more time to know how to write feedback on student's program"

"Code had to be compiled elsewhere"

How it could have been improved:

"How can we expect you to improve on perfection!!"

"The feedback applet would provide some interaction between the tutor and students."

"It always worked, bigger screen, can run progs, auto-flag errors, auto mark scripts etc."

"You could compile the programs with it"

Again, many of these problems were fixed for the following year.

Exam Results

To analyse the effectiveness of the system, we compared two groups of students, one group had used the system and the other group had not. We omitted students who were not in the Computer Applications course and for whom we had no Leaving Certificate data.

Year	**Number of students**	**Exam mean (Variance)**	**Leaving Points mean (Variance)**
2001 (control)	277	56.6 (22.9)	415.6 (43.5)
2002 (TutorBoard users)	124	55.2 (22.9)	396.8 (43.9)

Table 2: Comparison of student performance using/not using system

There was less than half the number of students in 2002 than in 2001. This was due to fewer applications to the course. The Leaving Certificate points requirements to qualify for the course dropped so that more places could be filled. This should only influence the results of our study negatively because the control group were more academically able.

The results were as good the year the system was used despite the fact that the ability of the students enrolled in the course was lower.

We focused our attention on the students who got low points in their Leaving Certificate. The Higher Education Authority [40] identifies these students as most likely to fail to complete their course of study. A comparison between these at risk students' results in Fall 2002 with their counterparts of the previous year shows a statistically significant improvement.

Figure 12 shows the average marks for the students in the various points range for 2001 (with no feedback system in place) and 2002 (using the system described in this thesis). The numbers above the bars represent the number of students in each category. Notice the improvement for students with entry points of 350 to 370 points. In 2001 the mean was 38%, while in 2002 it was 48%.

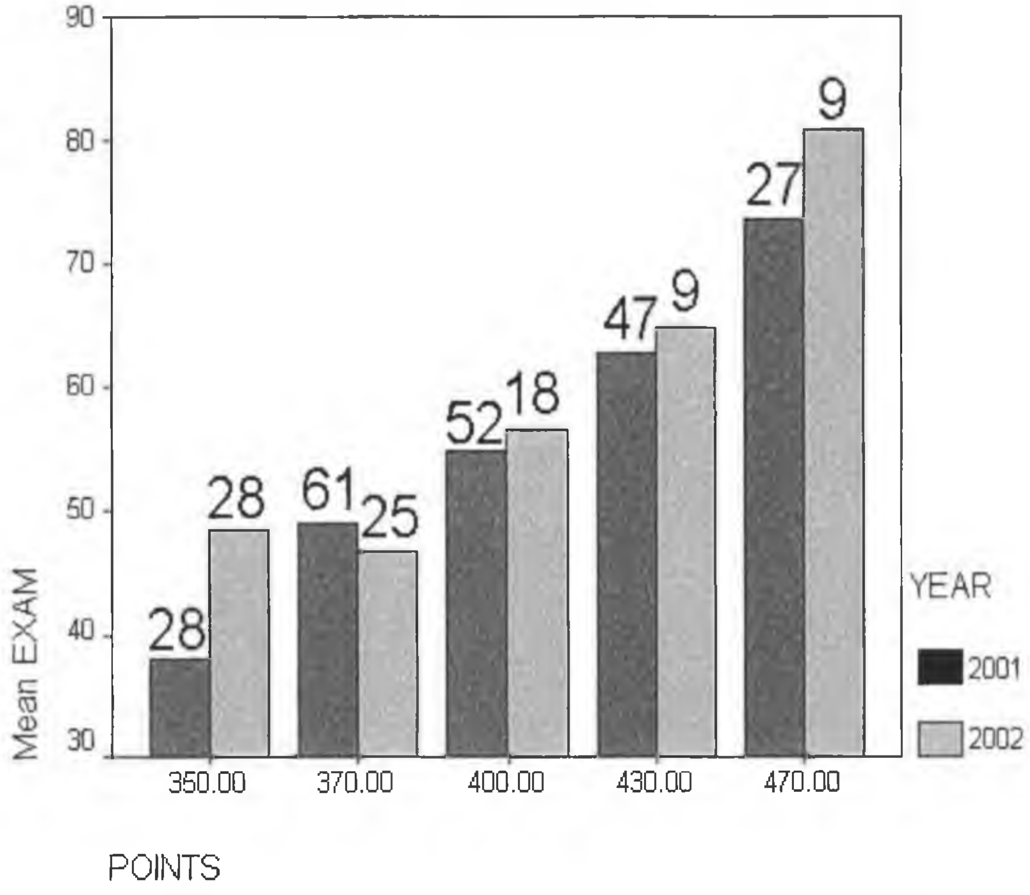


Figure 12: Comparison of exam results for 2002 and 2001 grouped into groups of equal points

In particular for the low points students (350 to 370), the t-test for equality of means produces a test statistic of t=-2.123. This is statistically significant. You would expect to see a t value like this by random chance only 4 times in 100. So, there is a statistically significant improvement in the at-risk low points group.

We examined the hypothesis that any particular tutor might have influenced their students' exam results. We compiled a table of how each tutor group's students averaged in the exam (see Table 3). Each tutor took a different approach to giving feedback. The ways they used feedback elements are also included in Table 3. There is no evidence that the exam marks depended on the type of feedback the tutors gave. Some tutors had only Mathematics students. The Mathematics students generally have higher entry points to the course. So it may have been the case that these tutors got better results because they had more academically able students.

Tutor	Java Sources Marked by tutor	Java Sources Submitted to tutor	Text Elements Used	Doodle Elements Used	Predefined Comments Used	Total Doodle Length	Logins	Average Mark
A	30	38	142	83	6	3255	22	41
B	34	38	98	43	2	2299	28	48
C	48	55	90	103	12	4943	38	49
D	27	42	32	25	7	1589	30	50
E	31	41	41	2	2	223	29	51
F	50	55	81	40	0	2400	37	52
G	36	36	84	209	0	7157	36	52
H	29	29	87	13	0	559	22	54
I	29	30	71	12	16	981	42	54
J	39	42	96	83	0	4190	32	56
K	23	36	45	101	0	3678	42	56
L	30	36	51	52	0	1591	22	57
M	57	59	76	58	3	3398	51	57
N	36	41	46	0	0	0	37	57
O	38	50	81	96	30	6216	34	58
P	31	34	111	27	0	563	23	60
Q	39	41	105	131	81	8394	34	60
R	58	66	114	86	1	3571	39	61
S	**47**	**48**	**210**	**197**	**26**	**7239**	**44**	**62**
T	35	42	119	12	4	1330	38	62
U	38	42	78	98	3	3398	30	62
V	37	53	107	101	31	5560	38	65
W	**23**	**38**	**35**	**17**	**0**	**772**	**41**	**66**

Table 3: Further analysis of tutor feedback and students exam results.

Indeed the tutor whose students got the highest marks used the feedback types sparingly, and didn't use the predefined function at all. Some tutors were able to express themselves with succinct feedback (it could be that these tutors were able to explain the material very well in the labs) while others expanded more on their explanations through the TutorBoard. Looking at the two highlighted lines shows two successful approaches to tutoring that are in sharp contrast. Tutor 'S' gave lots of feedback, while tutor 'W' was conservative with his comments. Both their groups of students received good marks. Further analysis failed to show a link between exam marks and the type or volume of feedback that the students received.

This is not surprising as students learn in different ways and tutors have individual styles of teaching. The TutorBoard system allowed the tutors to give as much or as little feedback as best suited their teaching style.

Conclusion

Feedback is an integral part of the learning process. Through personalised, fast, quality feedback we succeeded in increasing student motivation and confidence. The exam results show that less able students benefited most from the system; they gained a statistically significant improvement over their counterparts who did not use the system. This was perhaps because less able students find it discouraging when they struggle over problems that other students find trivially easy. Our web-based feedback system was able to provide individual attention in order to encourage the less able students to persevere with the task of learning how to program.

The TutorBoard system also went some way to solving the problem of increased class sizes. It was very useful in managing large volumes of student assignment submissions, handling 992 file submissions over the semester with most assignment submissions receiving feedback within 2 days.

The students' comments on the system were positive. Their comments show that feedback increases satisfaction and confidence, just as Keller theorises. From the questionnaires the most dissatisfied students were the ones who were dissatisfied because their tutor had not given them enough feedback. This indicates that the students themselves realise that it is important to get feedback.

Chapter 7: Motivating Peer- and Self-Assessment

Introduction

After the successful deployment of the feedback system it was modified and extended for the Introductory Programming course the following year. This time we wanted the students to have a look at the assessment process, examine other ways of doing the assignment (good and bad) and to take a deeper look at their own effort.

To this end, the TutorBoard system was extended to allow peer-assessment and self-assessment. After the students had submitted their assignment, the system guided them through the assessment process. They could carry it out in their own time. We required them to peer-assess at least one randomly selected assignment as well as self-assessing their own assignment. The student had access to the same tools that the tutors had to mark, comment on and grade the assignment. The very minimum required of the students was to supply a grade.

Autumn 2003

The feedback system was used with the Introductory Programming in the first semester, again the same course as the previous year. As in the previous year, the students had two lab sessions weekly on Tuesday afternoons and Thursday mornings, each lab was two hours long. They had a weekly assignment, which was due before the Tuesday, and they received their assignment back, marked by their tutors by the end of the week, through the web based TutorBoard feedback system. This time there were 20 tutors and 210 students. There were Computing students and Mathematics students in the class.

Analysis of Usage

Again the widespread usage of the TutorBoard system proved that the students wanted to read their feedback. The continued usage throughout the weekend (figure 13) and during the evenings (figure 14) shows that they were getting quicker access to feedback than they could have by the traditional way.

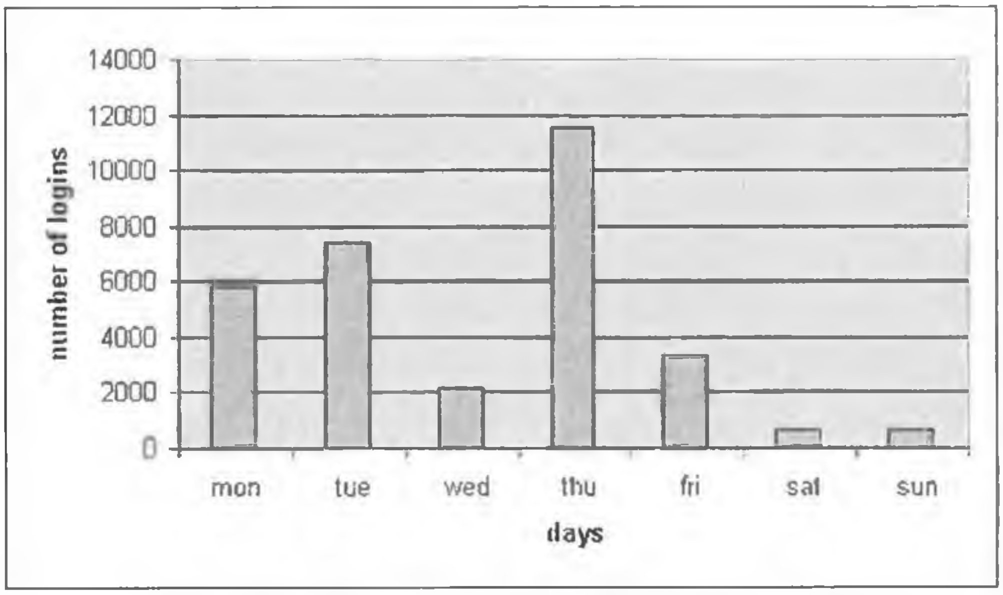


Figure 13: Total number of logins per day across the Semester 2003.

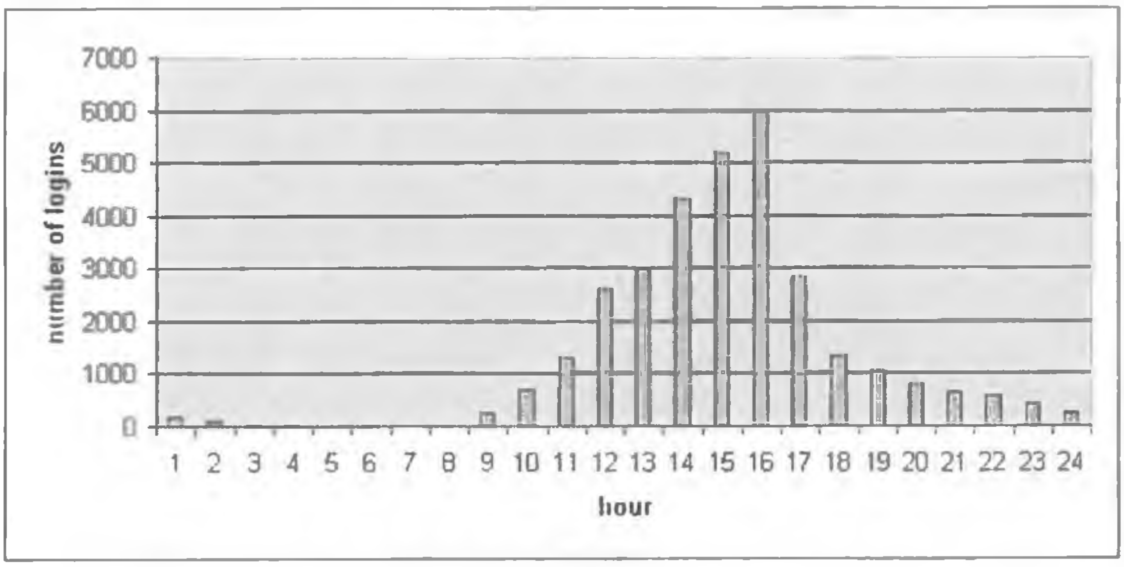


Figure 14: Total number of logins per hour across the Semester 2003.

The login numbers are almost 3 times what they were for the previous year. The system was used for peer and self-assessment this year so it changed the way that the students used the system. They were logging on more often to check if they had been peer assessed yet. Figure 15 shows there were a lot of requests for feedback, significantly more than the previous year.

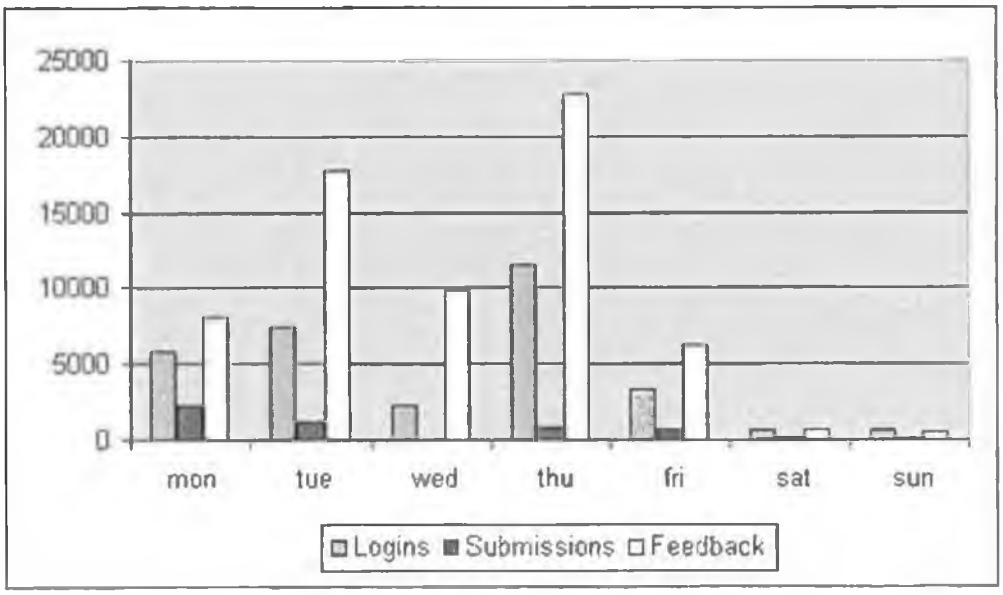


Figure 15: Further break down of usage of feedback system

Another reason that could have contributed to the larger logon numbers this year was our decision to run the server on Linux. Often at peak times the server was not able to handle the number of requests it received, resulting in students' sessions crashing, so they would have to log in again. The reason for this was a bug in the code that only surfaced when the system was run on Linux.

The TutorBoard system handled more files this year. The total number of files submitted was 3101 compared with 992 for the previous year. The total number of assignments submitted was 1468. The following table gives more data on the usage.

Java sources submitted	3101
Java sources marked	1259 by tutors (plus 3221 by peers)
Feedback Components	
Text elements	2682 by tutors (plus 985 by peers)
Scribble elements (number of pixels)	154658 by tutors (plus 90855 by peers)
Predefined comment elements	56 by tutors (plus 195 by peers)

Table 4: Table of feedback provided through TutorBoard in 2003.

All this data shows that the system was popular and thoroughly used. In addition this table shows that the use of peer assessment increased the amount of feedback the students received.

Motivating Peer-Asssessment and Self-Assessment

The purpose of this trial was to investigate how the system could be used for peer and self-assessment. In particular:

1) Did the system motivate the students to provide feedback to their peers and themselves? From the literature review we would expect this to be the case. Keller [27] states that students enjoy getting feedback. Lepper's [33] research says that motivation can come in the form of extrinsic factors or intrinsic factors. We offered both, extrinsic motivation coming from the fact that they received extra marks, and intrinsic motivation comes from the feedback.
2) What sort of feedback did they provide? The literature in this area says that students can give relevant feedback to their peers [17]. This is because they are learning to overcome the same problems at the same time. The problems and the solutions are still fresh in their minds. We analysed the sort of feedback to see if this was true and that the students were capable of producing good feedback.
3) Was this feedback useful and appreciated by their peers? This question is really an extension of the previous question. Before we asked if students can provide feedback on their peers. Now we ask if students found feedback from their peers useful. We expected this to be the case from [9], [8] and [48].
4) Did the process of peer-assessment and self-assessment help learning? We believed again that this would be the case from previous studies [54] and [34].

1) Motivating feedback

Bonus marks and the access to their feedback motivated the students to complete the process. 146 out of the 210 students registered on the system fully marked all the assignments that they submitted. That represents almost 70%, with the majority of the others marking all but one or two. Table 5 shows the number of students who submitted assignments and the number of students who fully marked an assignment per week. To aid comparison the percentage of students who fully marked their peer's and their own assignment is also shown. Overall it was consistently in the range of 87% to 92% except for the final week (when the students were busy revising for the end of semester exams). These numbers show that the motivating strategies were working.

Assignment Number	**Marked peer and self**	**Number of assignments submitted**	**Percentage marked**
#1	170	193	88%
#2	173	193	90%
#3	176	192	92%
#4	169	195	87%
#5	168	187	90%
#6	164	188	87%
#7	152	173	88%
#8	116	146	80%
Average	161	183	88%

Table 5: Percentage of weekly assignments marked.

These findings are further underlined through student's comments like the answer below to the question, what was the best thing about the system.

"the fact that you can get bonus marks!!"

The students' comments revealed a number of negative motivating factors that we can address in future.

"Trying to mark someone else's code and not being able to compile and test the code."

"Assessing peers - it is difficult for me to determine whether a program is correctly written without running it. It should be possible to copy and paste the code to check if it compiles and runs."

"Marking others was relatively difficult at best."

The students were not able to copy and paste the code from the TutorBoard but they were able to download the code to their own machines to compile them. The students were just looking for an extra convenience from the TutorBoard here.

2) Type of Feedback

Students tended to provide the minimum amount of feedback with a few exceptions. We counted the number of comments that each student made in his/her feedback. The maximum number marked was 16 assignments (that is 8 peer assessments plus 8 self assessments). However not all students fully marked all their assignments and furthermore not all students submitted an assignment weekly. We estimate that each student marked on average 75% of the assignments, or 12 assignments.

Total number of comments over the semester provided by students	**Number of students**
0	114 students
1 – 5	43 students
6 – 11	22 students
12 and higher	21 students

Table 6: Type of feedback produced

The above table shows that about half the class produced the absolute minimum feedback, providing nothing more than a grade to their peers and to their own assignment. Of those that did provide further feedback, it can only really be said that 21 of those made a satisfactory effort by providing at least one comment to each assignment (12 comments or above in total). There were a couple of notable exceptions, 6 students actually provided over 30 comments to their peers over the semester.

Perhaps the students resented our method of withholding their grade until they had completed the feedback process.

3) Was the feedback useful

The students were asked directly through the questionnaire if they found their peer's feedback useful. 36 of the students that replied agreed or strongly agreed that it was. Yet only 21 students made an effort at providing feedback. So it is likely that if more students provided good feedback then more of the class would have found their peer's feedback useful. So, it seems as if some students were able to teach their peers through their feedback.

This is how the students expressed it (when asked what the best thing about the system was):

"From time to time getting a marking from a peer that actually had something to do with the code and wasn't just an empty insult."

"It was good getting what other people thought of your programs"

4) Helping learning

We looked for evidence that the process of evaluating a peer helped learning. We hoped that seeing other code would teach the students:

1) to realise their own mistakes,

2) to understand what an assessor is looking for when evaluating a program and

3) to be able to distinguish between good and bad programs.

Table 7 shows the students responses to these questions on the questionnaire. The numbers of students agreeing with each of these statements are roughly equal but not large. So, it is not unreasonable to assume that the students making the effort benefited in these three aspects of learning equally.

	Agree			Disagree	
Marking my peers helped me to realise mistakes that I made in my own assignment	15	30	45	41	22
Marking my peers and my self helped me to understand the assessment process better	16	33	60	28	16
The process of marking some else's code helped me to understand what makes a good program	16	33	58	33	14

Table 7: Questionnaire answers

Many students commented on the learning benefits of evaluating a peer:

"Being able to read other student's code and seeing how they tackled the problem."

"It was interesting to see other programs and other ideas. Also helped to understand where marks were gathered and where I might be falling down."

Problems

The trial did not run completely smoothly. Some students just failed to understand the concept of the exercise:

"Annoying grading other peoples code (hard reading other peoples code)"

"Marking others was fairly pointless and more of a chore."

"I didn't like the fact that I had to mark my peers work in order to get my mark. most of the times I (and others) just gave 90% to all works."

And when asked, "What was the best thing about the system?" one answered:

"Witnessing the inferiority of my peers first hand".

The benefits and the purpose of peer assessment need to be explained to the students. Not explaining this clearly to the students was a mistake on our part.

Exam Results

Analysis of exam results shows that there is a clear correlation between exam results and the amount of feedback students provided. Using the written exam results as the dependant variable and the total number of comments provided by the students as the independent variable (see figure 16) produces a regression model that accounts for 6% of the variance in the exam marks ($R^2 = 0.061$). Running a t-test on the data to examine whether the exam results are totally unrelated to the amount of comments produces a value of $t = 3.586$, which is statistically significant.

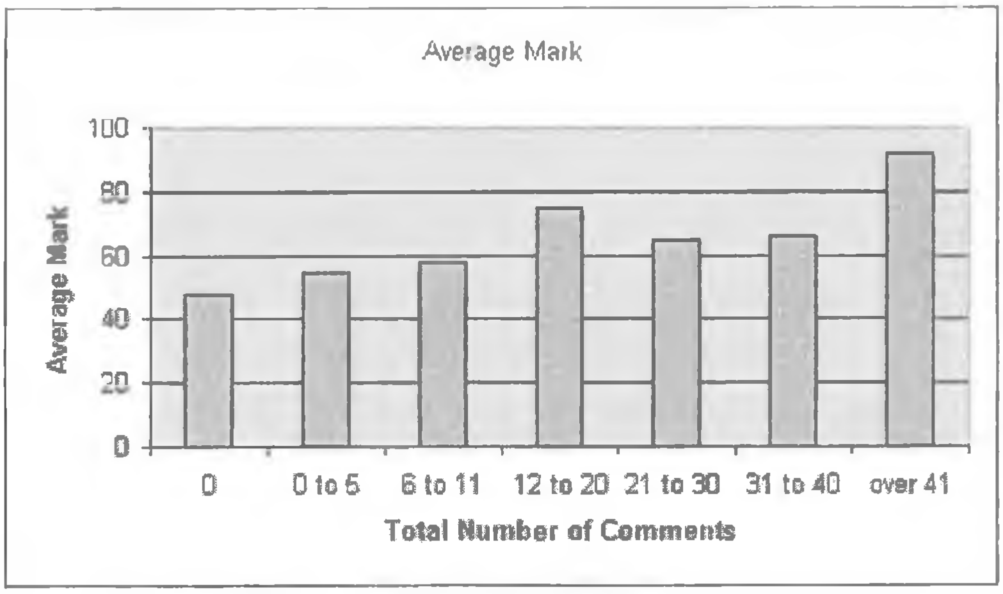


Figure 16: Exam Results depend on effort made to give feedback

Of course, this correlation could simply mean that the more academic students were more inclined to give feedback than the less academic students. It may have no affect on their grade at all. However if the smart students provide more feedback, perhaps we should be encouraging all the students to be more like them.

Conclusion

We have successfully developed a system that supports peer-assessment and self-assessment; the system was employed for teaching computer programming, but should have many more applications. The design of the system means that a lot of the normal administration disappears. So all the lecturer has to do is set up the assignments.

The system was successful in motivating the students to provide feedback on their peers as well. Each week about 88% of the students fully marked another peer and their own assignment. Perhaps some of this feedback was irrelevant, as some students gave 90% to each assignment just to get their own feedback. Self-assessment and peer-assessment did not seem to be beneficial for every one. About half the class did, however, learn from the exercise or otherwise benefit by getting extra feedback.

However, the fact remains that the majority of the students only provided minimal feedback. The reason for this seems to be that some students just do not believe that they benefit from this exercise. Different people learn in different ways. In future work we will encourage the students to provide peer- and self-assessment and describe to them why it is important.

Anecdotal evidence suggests that the tutors took the grading of the assignments more seriously. There was more pressure on them to provide a fair mark since the students' bonus mark depended on their mark being fair. This was one advantage of our approach. Another reason for the tutors to be interested in providing a fair mark is that their students now understand the assessment process better and would have more confidence in questioning their tutor. It would be interesting to look into this in more detail in future.

Chapter 8: Pair Programming

Introduction

For both years (2002 and 2003) that the TutorBoard system was used, the students were encouraged to use paired programming for their assignments. Their tutors explained the methodology of paired programming to them. The students were given the option to program alone but most decided to program with a partner. For example in Autumn 2003 we got 701 source files submitted by individuals and 796 source files submitted by pairs. This means that for every student working on his/her own there were at least 2 doing pair programming.

Generally the students were given an assignment at the beginning of the lab session on Thursday. The tutors encouraged students to pair up to complete the assignment. The students were generally free to choose their own partner. Usually they paired with someone else from the same tutor group. Some kept the same partner throughout the Semester and others changed partners as they wished. The pairs worked together in the lab, and could get help from their tutor if needed. The assignment then had to be submitted through the system before the next lab session. Both partners had to be present to submit the code.

Results

It has been speculated that pair programming creates a healthy pair pressure [60]. This pair pressure kept the students on task and encouraged them to complete their weekly assignments. 75% of the students completed and submitted their weekly lab assignment.

Pair programming also encouraged social interaction. Enjoyment and satisfaction are reasons for a student to values a course [27]. In this case the students enjoyed the social interaction. The students found the social interaction helpful and many singled it out as the greatest benefit to them. For example, in response to what the greatest benefit of paired programming one student commented that it was *"a good way to make friends in the beginning"*. The tutors also agreed with this point, that the social benefit of pair programming was useful. These social benefits also improve retention rates, as a happy student is less likely to drop out. McDowell found this to be the case [38] and our findings supports this. In Autumn 2002, only 7% of the students registered on the course dropped out before the end of the year exam, and in Autumn 2003 only 5% dropped out. These drop out rates are low in comparison to the national averages [40].

Student Reaction

The paired programming initiative evoked a vocal response from the students. Some students did not find it helpful at all. However, for the most part the students found it useful. In our survey at the end of semester 1 in 2002, 84 out of the 128 responses to our questionnaire said they enjoyed pair programming while only 20 said that they did not. In 2003, 92 out of the 141 said they enjoyed it while only 28 said that they did not. Mostly the answers to the questions do indicate that the students believed that the pair programming helped them code quicker, write neater algorithms, understand the code, and trust their code; they also agreed that they learned a lot from their partner.

For the tutors there was also the benefit that the students demanded less time when they were pair programming. This was observed when walking through the labs when a paired lab was in progress. In the non-paired labs the tutors were usually busy instructing their students. In the paired labs they had more time to read over and mark the students' assignments.

Getting the Right Match

We found that for the pair programming to be more successful that it would be important to try to match up the students based on their abilities, or their personalities [26]. Both students and tutors suggested this in their comments. We found the problems created by having students of different abilities and different personalities in a partnership could be categorised in three ways.

The first is when one student took over the keyboard and either did not give his/her partner a fair chance. In this case the less assertive partner lost out and did not get practice at applying their skills.

The following comments are indicative:

"Not a good idea, one partner usually does most of the work"

"My partner found programming very difficult. Sometimes it worked out that I wrote the code and submitted it under both our names, which didn't teach her anything."

"my partner was very quiet!"

"my partner was miles ahead of me in programming ability and was not patient enough to allow me to get the hang of it. he just did all the work without saying much,

i learn better when i have to solve a problem for myself as opposed to just being shown how to do something."

The second is similar but this time instead of doing the assignment without any help, the stronger partner is very patient and waits for his less able partner. It is possible that the stronger partner spends a lot of his time teaching rather than advancing to more difficult problems. Hence, his partner holds him back.

"It was good if the programmers were at the same level, they would learn from each other. My group I was better than partner and he got more out of it than me."

The third problem and final problem we focus on is that some students just do not want to program as a pair, they are used to programming alone. It is going to be hard to find a good match for these students.

"pair programming may be good but ive ended on my own for too long - i couldnt get into it, it didnt work."

The tutors attempted to work around these problems. They agreed that matching students is very important. The majority agreed that students should be paired up by equal ability.

"Seems like a good idea in theory, I would have hated it though! I think its essential to match the pairs off ie personality skill level etc for it to work at all."
"Sometimes, some stronger students seemed to take over. Whereas the weaker students were happy to let their partner do the exercises."

In contrast to the above tutors' approach, at least one of the tutors did find it useful to pair up weak students with stronger students. The less able students were more open to suggestion from their own classmates and so learned better.

One student summed it up neatly in the questionnaire, it is *"hard to find the right partner, that's life!"*.

Conclusion

We found that paired programming was useful for the following reasons:

- It helps the students to settle into their course.
- It is likely that it contributes to reducing drop out rates.
- It also frees up some of the tutor's time giving them more time to prepare for the next lab or mark assignments if they wish.

However the students and tutors raised many concerns about pair programming in the questionnaires. Most importantly, how do you match up the students? By matching them up by similar ability the less able pairs will fall behind because they are not as strong at programming. By matching less able students with strong students, there is a danger that the stronger student will not wait for them. In addition, when matching up students you must take into account their personalities. Maybe it is best to rotate partners every lab. In that way students make lots more friends – which can be an important factor in reducing drop out rates [40].

However, it is an important skill to be able to work co-operatively with a partner or in team. In the students' future careers as developers they will not always get to choose their partners or colleagues. Sometimes they will have to work with people they do not get on with. So, if we get the match wrong in the lab, then the student will receive some experience of working with people that have different personalise as they do.

Chapter 9: Conclusion

There is a problem with the current methods used to teach Computer Programming [40], [37]. Students are not getting enough quality feedback on their assignments in DCU. It is important that they get as much individual feedback as possible. Large class sizes and limited time available by the lecturer add to this problem. We have addressed this problem of inadequate feedback by developing a system that allows tutors and students to give personal feedback to each other. We carried out a critical evaluation of the system.

Tutorboard gave the students extensive feedback. Instead of receiving short feedback once or twice per semester the students received feedback every week. The feedback was available instantly after the assignment was marked – so the student could review the feedback while the assignment was still fresh in his mind. The improvement is well summed up be the lecturer's delight when he commented, "It's great. 200 plus students get feedback on their assignment every week within a couple of days of the assignment."

The web-based nature of the feedback was very useful. As computing students, the students would have been comfortable using this medium. It also meant that they could have access to their feedback from anywhere and at any time. This gives the students the flexibility to learn at their own pace. Questionnaires and comments show that the students enjoyed using the system. They logged on early and often to access their feedback.

The feedback was very clear and yet simple. The idea was to mimic the familiar type of feedback that students have seen right from the first day of school. The feedback appeared on the computer screen as red marks just as if the tutor had scribbled his comments over the assignments with a pen. The tutor could draw diagrams to illustrate a point, make comments and link those comments to a specific line of code, or to a specific block of code.

Tutorboard facilitated the provision of personal feedback to the students. Unlike other forms of web-based or automated feedback it was not cold and impersonal. Each tutor had a small group of students to supervise. They could get to know each one. They could get to know each student's motivation; get to know their weaknesses and their strengths. With this in mind the feedback would be individualised for each student. This is akin to the Socratic way of education.

As well as tutor feedback the students received feedback from their peers. Their classmates often will have a better understanding of the difficulties that they are facing and so their extra feedback will be useful. Tutorboard made this process easy. The student was first directed to assess one of their peers. This exercise will put them in a critical mindset. The next step is to evaluate their own assignment. It is hoped that the marking schemes, the sample answers, looking at other student's attempts and being in a more critical frame of mind will help the students to learn something new from looking over their assignment again. After finishing the assessment process the student has access to any feedback they received from peers and from their tutor.

The strategy to motivate peer/self assessment did work. 88% of students peer and self assessed the assignments fully each week. However, we did find that some students disliked the idea of withholding feedback. We can address this in future by giving the students more freedom – so that they need only take part in the peer/self assessment if they wish. They should get their tutor's feedback regardless.

The majority of the students also enjoyed pair programming. Having a partner to help with the assignment took pressure off the students. It helped them to make friends and settle into their course. They learnt to work as part of a team.

Tutorboard contributes to each of Keller's four conditions to motivate students [27].

<u>Attention</u> – the students receive extra attention from the tutors, from the peer assessment and from pair programming.

<u>Relevance</u> – the student can see the relevance in the course particularly through pair programming and individual instruction.

<u>Confidence</u> – encouraging feedback increases confidence. Also working closely with the tutors the students see how much they can learn by the end of the year. The tutors are only 1 year ahead of the students. This gives the students confidence.

<u>Satisfaction</u> – students enjoyed receiving feedback. Students enjoyed pair programming.

It is not surprising then that the average grade improved when TutorBoard was used. The improvement in the less able student's grades was statistically significant.

There were many clear benefits for the tutors too.

Benefits

The tutors could correct assignments quickly and easily without having to sort through email attachments, floppy disks or paper printouts. All that they needed to manage their small group of students was available over the web. They found the TutorBoard useful for correcting all sorts of errors that the students made. During the system evaluation over 200 students received feedback on a weekly basis within 2 or 3 days of submitting the assignment.

The tutors' feedback and marks were important. The tutors realised that their marks needed to be accurate as the students' bonus peer assessment marks depended on the tutors' mark. The students learn about the marking criteria through peer assessment and so are better armed to question the tutors' feedback. Therefore the tutors would put more thought into the feedback and grades. They would need to be confident that they could defend the grade.

Pair programming and self/peer assessment were found to be effective in providing students with additional feedback and involved no extra work for the tutor. The tutors reported that in the pair programming labs the students were able to help themselves. A lot of the straightforward problems could be solved between the pair. The tutors could spend their time on answering more involved questions, or marking assignments, or preparing lessons for the next lab.

Educators have also expressed an interest in the tool. One of the reviewers of our paper for the ITiSCE (Innovation and Technology in Computer Science Education) 2004 conference in Leeds said that they would be interested in using such a tool even though class sizes at the university that they taught at are much smaller. There were a number of different educators from a number of different universities across the world there.

References

[1] Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*, 89, 369-406.

[2] Apache Tomcat [Online] <http://jakarta.apache.org/tomcat/index.html> (last accessed 2/6/'06)

[3] Audi, R. 1994. *On the Ethics of Teaching and the Ideals of Learning.* Academe, 80(5), pp27-36.

[4] Bancroft P., Roe. P. 2006. *Program Annotations: Feedback for Students Learning to Program. Proceedings of the 8th Australasian Computing Education Conference.*

[5] Barnes, C. More programmers going "Extreme". CNet [Online] <http://news.com.com/2100-1040-255167.html> (last accessed 2/6/'06)

[6] Beck S. *How Socarates taught.* [online] <http://www.san.beck.org/SOCRATES3-How.html> (last accessed 2/6/'06)

[7] Bellis, M. *The Dawn of Windows* [Online] <http://inventors.about.com/library/weekly/aa080499.htm> (last accessed 2/6/'06)

[8] Bhalerao, A. and Ward. A. 2001. *Towards electronically assisted peer assessment: a case study. Alt-J--Assoc. Learning Technol. J.* 9, 1, pp26-37.

[9] Biggs. J. 1999. *Teaching for Quality Learning at University.* Buckingham: SRHE and Open University Press

[10] BlackBoard [Online], <http://www.blackboard.com> (last accessed 2/6/'06)

[11] Bloom, B.S. (Ed.) 1956. Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain. New York ; Toronto: Longmans, Green.

[12] Brown, G., Bull, J., Pendlebury, M. 1997. *Assessing student learning in higher Education.* Routledge, London, 170-184.

[13] Cockburn, A. and Williams, L. 2001. *The Costs and Benefits of Pair Programming in Extreme Programming Examined.* Addison Wesley, 2001.

[14] Creative Technology – Software for Teaching – Markin [Online]. Available from: <http://www.cict.co.uk/software/markin/index.htm> (last accessed 2/6/'06)

[15] Daly, C., Donnellan, D., Ward, M., Walshe, R. 2002. *The Lab Tutor System of a Large Undergraduate Class: The Lab Tutors' Perspective. Proceedings joint SEDA and AISHE conference, Dublin Castle*, 2002.

[16] Dochy, F., McDowell, L. 1997. *Assessment as a tool for learning.* Studies in Educational Evaluation, 23, pp279-98.

[17*] Fox, S., MacKeogh, K. 2003. Can eLearning Promote Higher-Order Learning without Tutor Overload*? Open Learning Vol.18, No. 2.

[18] Hightower. R, Lesiecki N. 2002. *Java Tools for eXtreme Programming.* p5.

[19] Hughes, M. Draw the world: Create networked whiteboards with Java 1.1 [Online] <http://www.javaworld.com/javaworld/jw-11-1997/jw-11-step.html> (last accessed 2/6/'06)

[20] IBM, Design Basics [Online] <http://www-306.ibm.com/ibm/easy/eou_ext.nsf/publish/6> (last accessed 2/6/'06)

[21] Jackson, D. 2000. *A Semi-Automated approach to online assessment.* ITiCSE 2000.

[22] Java BluePrints: Guidelines, Patterns, and code for end-to-end applications [Online] <http://java.sun.com/blueprints/index.html>

[23] Java Technology: The early years (1998) [Online]

<http://java.sun.com/features/1998/05/birthday.html> (last accessed 2/6/'06)

[24] Jeffries, R. 2001. *What is Extreme Programming?* [Online].

<http://www.xprogramming.com/xpmag/whatisxp.htm> (last accessed 2/6/'06)

[25] Jenkins, T. *Teaching Programming – A Journey from Teacher to Motivator. Proceedings of the 2nd Annual Conference of LTNS Centre for Information and Computer Sciences.*

[26] Katira, N., Williams, L., Wiebe, E., Miller, C., Balik, S., Gehringer, E. 2004. *On understanding compatibility of student pair programmers. Proceedings of the 35rd SIGCSE technical symposium on Computer science education*, ACM Press, 2004, pp. 7-11.

[27] Keller, J. M. 1987. *Development and use of the ARCS model of instructional design. Journal of Instructional Development*, 1987, 10 (3), 2-10.

[28] Keller, J.M. and Keller B.H. 1989. *Motivational delivery checklist, Florida State University.*

[29] Kolb, D. A. (1984). *Experiential learning: Experience as the source of learning and development*. Englewood Cliffs, NJ: Prenticve Hall. p. 42.

[30] Krishna K. Adusumilli, Bassem Al-Halabi, Sam Hsu. 2000. *Softboard – A Web-based Application Sharing System for Distance Education. IN: The International Conference on Information Technology: Coding and Computing (ITCC'00)*. pp338-343.

[31] Kulik, J.A. and Kulik, C.L.C. (1988) Timing of feedback and verbal learning. *Review in Educational Research*, 58(1), 79-97.

[32] Lawhead, P. 2003. Legos, *Java and Programming assignments for CS1. Proceedings of the 34th SIGCSE technical symposium on Computer science education*, ACM Press, 2003, pp. 47-48.

[33] Lepper, M.R., Green, D., and Nisbett, R.E. (1973). Undermining children's intrinsic interest with extrinsic reward: A test of the "overjustification" hypothesis. *Journal of Personality and Social Psychology*, *28*, 129-137.

[34] Lewis, S. and Davies, P. 2004. *The automated Peer-Assisted Assessment of Programming Skills, JICC 2004.*

[35] Linux Online - About the Linux Operating System [Online] <http://www.linux.org/info/> (last accessed 2/6/'06)

[36] Matthew son of Alphaeus (ca. 62 AD), *Matthew's Gospel Chapter 7 verse 1, The Bible.*

[37] McCracken, M et al. 2001. *An international multi-institutional study of introductory programming courses. Report by the ITiSCE 2001 Working Group on Assessment of Programming Skills of First-year CS students.*

[38] McDowell, C., Werner, L., Bullock, H. & Fernald, J. 2000. *The effects of pair-programming on performance in an introductory programming course. ACM SIGCSE Bulletin*, 34(1), (March 2002) p. 38-42.

[39] Moodle - A Free, Open Source Course Management System for Online Learning [Online], <http://moodle.org> (last accessed 2/6/'06)

[40] Morgan, M., Flanagan R. and Kellaghan, T. 2001. *A Study of Non-Completion in Undergraduate University Courses.* Publication of Higher Education Authority of Ireland report. [Available online] <http://www.hea.ie/projects/completion/newsrelease28-05.htm> (last accessed 11/1/'04)

[41] Murray R., 1988. *Automatic Program Debugging for Intelligent Tutoring Systems.rogram.* Morgan Kaufmann Publishers.

[42] Mysql [Online] <http://www.mysql.com/> (last accessed 2/6/'06)

[43] Nawrocki, J. and Wojciechowski, A. 2001 *Experimental Evaluation of Pair Programming. Proceedings of the 12th European Software Control and Metrics Conference, ESCOM 2001*, (2-4 April 2001, London), 269-276.

[44] Netcraft: Webserver Survey Archives [Online] <http://news.netcraft.com/archives/web_server_survey.html> (last accessed 2/6/'06)

[45] Notes on using Java applets in browsers (2003) [Online] <http://www.ssec.wisc.edu/~tomw/jvm.html> (last accessed 2/6/'06)

[46] Odekirk-Hash, E., Zachery, J. 2001. *Automated feeback on Programs means students need less help from teachers.* SIGSCE 2001.

[47] Oman, P. *1990. A taxonomy for programming style.* ACM.

[48] Race, P. 2001. *A briefing on self peer and group assessment.* York, LTSN.

[49] Race, P. 1993. *Never mind the teaching – feel the learning, SEDA paper 80*, 1993, p. 17.

[50] Reges, S. 1988. *The Effective Use of Undergraduates to Staff Large Introductory Courses in Proceedings of the nineteenth SIGCSE technical symposium on Computer science education.*

[51] Rodger, S. 2002. *Introducing Computer Science Through Animation and Virtual Worlds. Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, ACM Press, 2002, pp. 186-190.

[52] Rory's Java Programming - Sun, Java and Microsoft [Online] <http://www.rorysjava.intelynx.net/sunjavams.html> (last accessed 2/6/'06)

[53] Saljo, R. 1976. *Learning in the learners perspective IV: Considering ones own strategy*, Higher Education Volume II.

[54] Sitthiworachart, J & Joy, M. 2003. *Deepening Computer Programming Skills by Using Web-Based Peer Assessment. 4thAnnual LTSN-ICS Conference*, NUI Galway.

[55] Somervell, H., "Issues in assessment, enterprise and higher education: the case for self-, peer and collaborative assessment", *Assessment and Evaluation in Higher Education*, 18, 221-233, 1993

[56] "Report and Recommendations of the task force on the Physical Sciences". [Online] <http://odtl.dcu.ie/mirror/irlgov/educ/sciencetaskforce-rpt.pdf> (last accessed 2/6/'06)

[57] Topping, K., "Peer assessment between students in colleges and universities", *Review of Educational Research,* 68, 249-276, 1998

[58] Waldo, R. 1980. *The teaching of documentation and good programming style in a liberal arts computer science program ACM.*

[59] WebCT.com [Online], <http://www.webct.com> (last accessed 2/6/'06)

[60] Williams L., Kessler R. 2003. *The Effects of "Pair-Pressure" and "Pair-Learning" on Software Engineering. Proceedings of the 34th SIGCSE technical symposium on Computer science education*, ACM Press, 2003, pp. 123-133.

[61] Williams, L.A. and Kessler, R.R 2000. *The Effects of 'Pair-Pressure' and 'Pair-Learning' on Software Engineering Education.* Proceedings of Thirteenth Conference on Software Engineering Education and Training, pages 59-65.

[62] Williams L., Upchurch R. L. 2001. *In Support of student pair-programming. Proceedings of the 32nd SIGCSE technical symposium on Computer science education*, ACM Press, 2001, pp. 327-331.

[63] Windows Products and Technologies History (2002) [Online] <http://www.microsoft.com/windows/WinHistoryDesktop.mspx> (last accessed 2/6/'06)

Appendix A

Java had many desirable properties that prompted us to use it as our main language of development. Many of these desirable properties are highlighted in the following technical anecdote.

Linux will not allow root access to its (graphics) X Server, and neither will it allow remote programs access to it. This is because the X Server is a server in its own right and so is vulnerable to attacks. The natural way to save the feedback was to serialize it using Java's serializable keyword and then to write it to the file system. The process of serializing meant that we had to serialize every comment, and every doodle. Since the comments were saved as Java Font objects serializing these was a problem. When Java is instantiating Font objects part of the process involves connecting to the graphics server or retrieving a graphics context. Now this is where we ran into a problem since Linux would not allow access.

The simple solution to this problem, as you will see, illustrates well the flexibility of Java, and its platform independence, which were two of the reasons we chose it as our main development language. The solution was to save the Fonts as String objects as opposed to Font objects. This meant that we also had to save the height of the fonts, and the font face separately. Now that we were not using any of Java's graphics objects to save the feedback Linux was happy to let us serialize the feedback and write it to the file system. This was all very straightforward with Java, we just switched one class with another.

Working on Windows

```
public class TextElement

{

    Font font;

    Rectangle bounds;

    int x, y, font_height;

    Color color;

    Vector lines;


    TextElement (Vector t, int x, int y, Color c, Rectangle b,

int h, Font f)

    {

        time = System.currentTimeMillis();

        bounds = b;

        this.x = x;

        this.y = y;

        lines = t;

        color = c;

        font_height = h;

        font = f;

    }


    ...


    public void paint (Graphics g)

    {

        g.setFont(font);

        g.setColor(color);

        Iterator i = lines.iterator();

        int line = 0;
        while (i.hasNext())
        {
            g.drawString((String) i.next(), x, y +
font_height*line);
          line ++;
        }
    }
}
```

Modifications for Linux

```
public class TextElement
{
    String font_name;
    Rectangle bounds;
    int x, y, font_height;
    Color color;
    Vector lines;

    TextElement (Vector t, int x, int y, Color c, Rectangle b,
int h, String f)
    {
        time = System.currentTimeMillis();
        bounds = b;
        this.x = x;
        this.y = y;
        lines = t;
        color = c;
        font_height = h;
        font_name = f;
    }
    ...


  public void paint (Graphics g)
  {
    Font f = new Font(font_name, Font.PLAIN, font_height);
    g.setFont(f);
    g.setColor(color);
    Iterator i = lines.iterator();
    int line = 0;
    while (i.hasNext())
    {
      g.drawString((String) i.next(), x, y + font_height*line);
      line ++;
    }
  }
}
```

Appendix B

The System (technical overview)

We can now sum up how all these technologies were put together to form our TutorBoard. The Java code was 11,397 lines long and contained 58 classes. Figure 17 shows a brief schematic overview of the system.

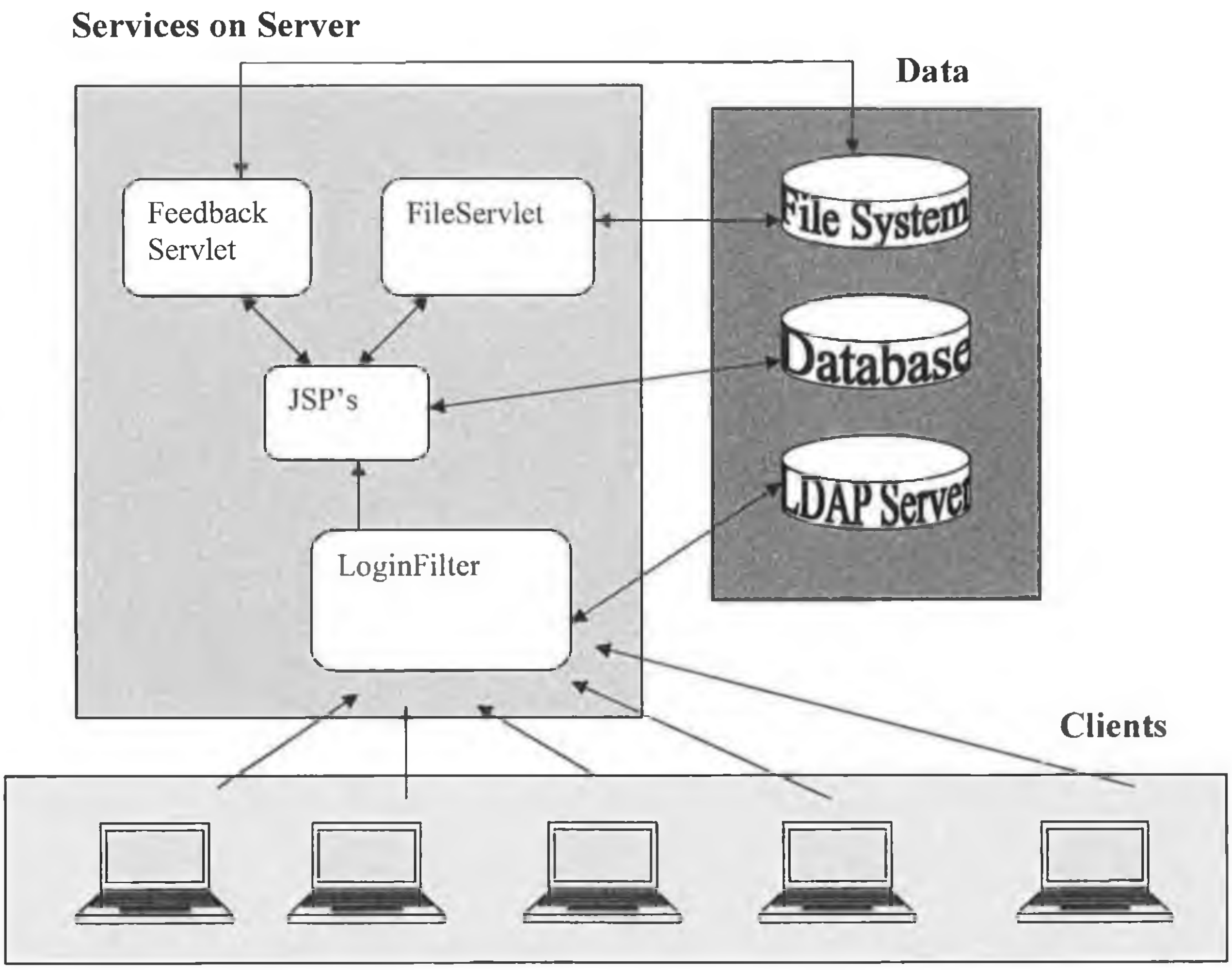


Figure 17: Client machines access the server via their web browsers. The server manages the login sessions and supplies all the feedback and files. It accesses data from a database, our file system and an LDAP server.

The 'TutorApplet'

The client side of the system was a simple whiteboard Java applet with a couple of added features. The bare bones of the whiteboard were already written and the code can be found on www.javaworld.com [19]. We re-used and extended this code to develop our 'TutorApplet'.

The first thing that was added to the TutorApplet was the ability to display java source code in the background. This task again illustrates very well the power of Java and why we chose it as our main development language.

It was made simple by java's comprehensive existing libraries and packages. There is a library distributed with the Java Standard Edition, which includes JEditorPane. The java JEditorPane is a java class that extends a Frame that includes simple methods for rendering HTML. Although its fairly recently developed, clunky and still has a couple of bugs it served our simple purpose very well. We also located a package very easily that converted java source code to HTML

Integrating the converted HTML code into the Javaworld whiteboard application was simple because Java is a highly re-usable language. All we needed to do was

- import the JEditorPane class in the whiteboard class.
- change Javaworld's WBContainer class to extend JEditorPane instead of Container. This is possible since JEditorPane is derived from Container and so WBContainer will lose none of its functionality.
- write a simple method that we can call that displays a URL in the background whenever the feedback changes.

Javaworld WhiteBoard

```
package org.merlin.step.nov;


import java.awt.*;

import java.util.*;


public class WBContainer extends Container implements
UpdateListener
{
   ObservableList elements;


   ...
}
```

Our TutorApplet

```
package ie.dcu.compapp.david.applet;


import java.awt.*;

import java.util.*;


import javax.swing.JEditorPane;


public class WBContainer extends JEditorPane implements
UpdateListener
{
   ObservableList elements;
   ...

   public void setTextChangeURL()
   {
      try
      {
         setPage(elements.getBackground());
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

Now we had a working whiteboard with syntax highlighted java code displayed in the background. We needed suitable tools for the whiteboard for marking or annotating the java source. The whiteboard already had a couple of tools, i.e. a box tool (that drew boxes) and a moving tool (that could move the boxes). In order to make the feedback worthwhile we developed a few more tools.

All these tools could be developed quickly and easily due to the use of interfaces in Java. An interface allows the development of pluggable components. All the new tools had to do was to implement a 'Tool' interface and they would slot into the rest of the program seamlessly.

The following diagram shows the design of the 'TutorApplet'. The diagram shows that the 'TutorApplet' can use many 'Tools'. For example 'TextTool' and 'DoodleTool' implement the Tool's interface.

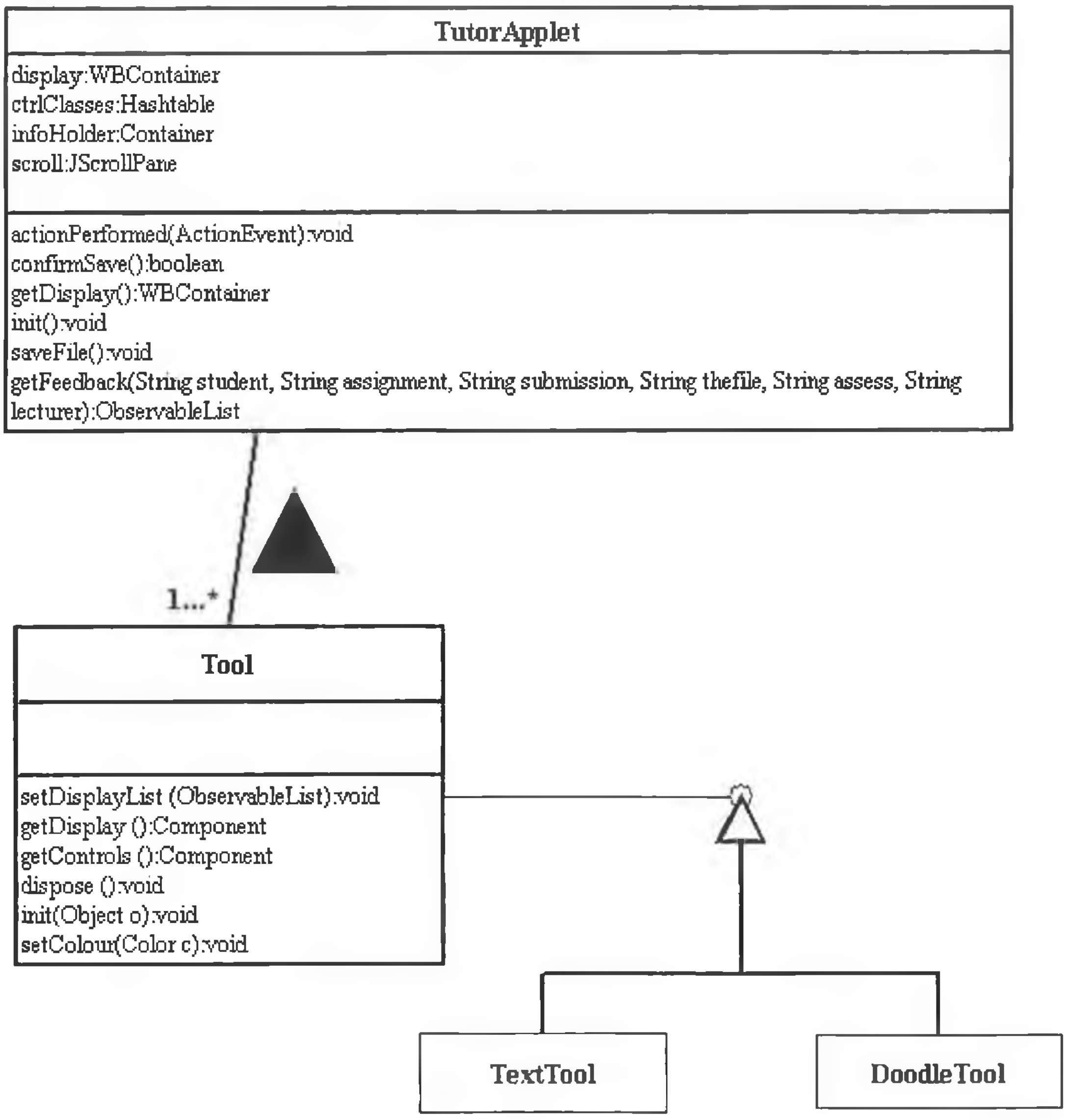


Figure 18: A UML class diagram describing the methods of the TutorApplet and showing the pluggable Tool interface design.

There were just a couple of other small additions that we made to the freely available javaworld whiteboard. A drop down menu was added that allowed the tutors to manage the files that the students had submitted. We also added an extra box to record a mark for the student's assignment. Finally there was a save button added to enable the tutor to save the feedback file.

The save utility was handy as it saved the feedback separate from the source file. In that way the original file was unchanged. This facilitated a couple of important features. It allowed students to turn on and off feedback. It also allowed many people to mark the same file.

In order to access the feedback the student used the same applet with a few restrictions. They were not allowed to write over their files, or save their feedback. They could however choose the different files from the menu and it would display the appropriate feedback. Also if the file was marked by a number of different people, as it was for peer marking then there were simple buttons that the student could use to turn off and on the different feedbacks.

The LoginFilter, Servlets, and JSP's

The marking and reviewing features were facilitated by a powerful backend server built on the latest Java Servlet and JSP technology, connected to a MySQL database and the University's LDAP database for student authentication.

All authentication was carried out by querying the University LDAP server. LDAP is a Lightweight Directory Application Protocol. It stores information on computer network users including email addresses, names, identity numbers and their organisational unit in a directory based structure. Every request that our system received had to be authenticated with the LDAP server. To do this we used a Servlet Filter. Basically a Servlet Filter can be configured to intercept every request before sending it on to the Servlet to be dealt with properly. Our Servlet Filter was very simple. It intercepted the request. If the user had not already been authenticated it would not pass on the request. Instead it would ask for a username and password, which it could authenticate via the LDAP server. If authentication succeeded a cookie could be set to indicate that the user was authenticated, and subsequent requests would be passed straight on by the Filter without doing any further processing.

Since authentication was done using the LDAP server we had the convenience of not having to deal with students lost passwords, or changed passwords. The student had a common password given to them at registration that they used to access all university computer resources including our system. It made it simpler for them, and simpler for us.

Once authenticated, the system pulled down information on the students from the MySQL database. A system of Servlets and JSP's also queried the file system to see what files that the student had already submitted and what feedback files that had been submitted too. It was then able to handle all the students' requests and display a list of their files submitted, display their feedback, show the files etc.

Communications Between Client and Server

Communication between the Servlets and the applet was another important consideration. We could deliver some information to the applet by passing parameters to it via the applet "param" tag. In this way the name of the student, his tutor, the name of the assignment, the names of the files submitted and some more information was passed to the applet.

However the applet had to download the source files and the feedback files to display to the users, so we needed a more sophisticated mode of communication. We developed a FileServlet that handled all requests for the source files and a FeedbackServlet that handled all feedback requests. The applet could communicate directly with these Servlets. The FeedbackServlet opened up a socket to the applet to send down the feedback, when there was a request for feedback. The main concern here was that the students could not download someone else's file and copy it. This was achievable because through the authentication process we knew exactly who was logged on and the system could work out whether this user should be able to access the requested file.

Security restrictions

Since a java applet is an application running on the client's machine there are certain restriction to prevent a malicious applet from doing any damage. File access is restricted and socket I/O is restricted. This was not a problem for writing the feedback Objects back to the Servlet because applets do allow you to open a socket back to the host that the applet was downloaded from, but not to another machine.

It was a problem for allowing the tutors to run the students assignments online though. In order to run the assignment online we had to redirect the standard out and standard in. So that now the standard out would print to the applets graphics area, and standard in would read from the keyboard. This sort of operation is not allowed because of the applet security model. Our solution is to have the TutorBoard sign the 'AssignmentRunner'. Users have the option of allowing signed applets to execute privileged actions, such as redirecting standard in and standard out.

Appendix C

Student Questionnaire

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
The TutorBoard system was easy to use					
The TutorBoard system made it possible to get feedback on your code without much delay					
The tutor feedback was clear and easy to read					
The tutor feedback gave me more confidence in my coding					
I found it useful that you could submit the code from anywhere at any time					
It was easy to submit code using the TutorBoard system					
I learnt alot from marking other students assignments					
I found feedback I got from my peers very useful					
Marking my peers helped me to realise mistakes that I made in my own assignment					
Marking my peers and my self helped me to understand the assessment process better					
The process of marking some else's code helped me to understand what makes a good program					
I didn't like having to mark myself					
I didn't like having to mark my peers					
The best feature of the TutorBoard system was ?					
The worst thing of the TutorBoard applet was ?					
I enjoyed programming in a pair					
I seemed to code faster with a partner					
In pair we could come up with faster and simpler algorithms than alone					
I learnt alot from my partner					
I understood the code better when I had to explain it to my partner					
I had more confidence in my code when programming in a pair					

My partner did most of the work

My partner slowed me down

The tutors knew the material well

The tutors were able to explain material very well

Please feel free to leave any general comments about the feedback system ?

Tutor Questionnaire

PART I (DID YOU LIKE THE FEEDBACK SYSTEM)

	Strongly Agree			Strongly Disagree	
The TutorBoard system was easy to use from a tutor's point of view.					
The TutorBoard system made it possible to mark assignments quickly					
It was easy to communicate syntax errors to the students with the TutorBoard system					
It was easy to communicate logical errors in the students code using the TutorBoard system					
It was easy to show the students good programming practices with the TutorBoard system (indentation, good variable names, short methods etc.)					
I found it useful to be able to provide the feedback whenever I liked					

- The best feature of the TutorBoard system was ...

- The worst thing about the TutorBoard system was ...

- The TutorBoard system could have been improved if ...

PART II (DID THE PAIR PROGRAMMING WORK)

	Strong ly Agree			Strongly Disagree	
The students co-operated well in the paired assignments					
The students seemed to code faster in pairs					
The students algorithms were usually faster and simpler when they programmed in pairs					
Students demanded less of the tutors time when they worked in pairs					

PART III (DID YOU LEARN ANYTHING FROM TUTORING)

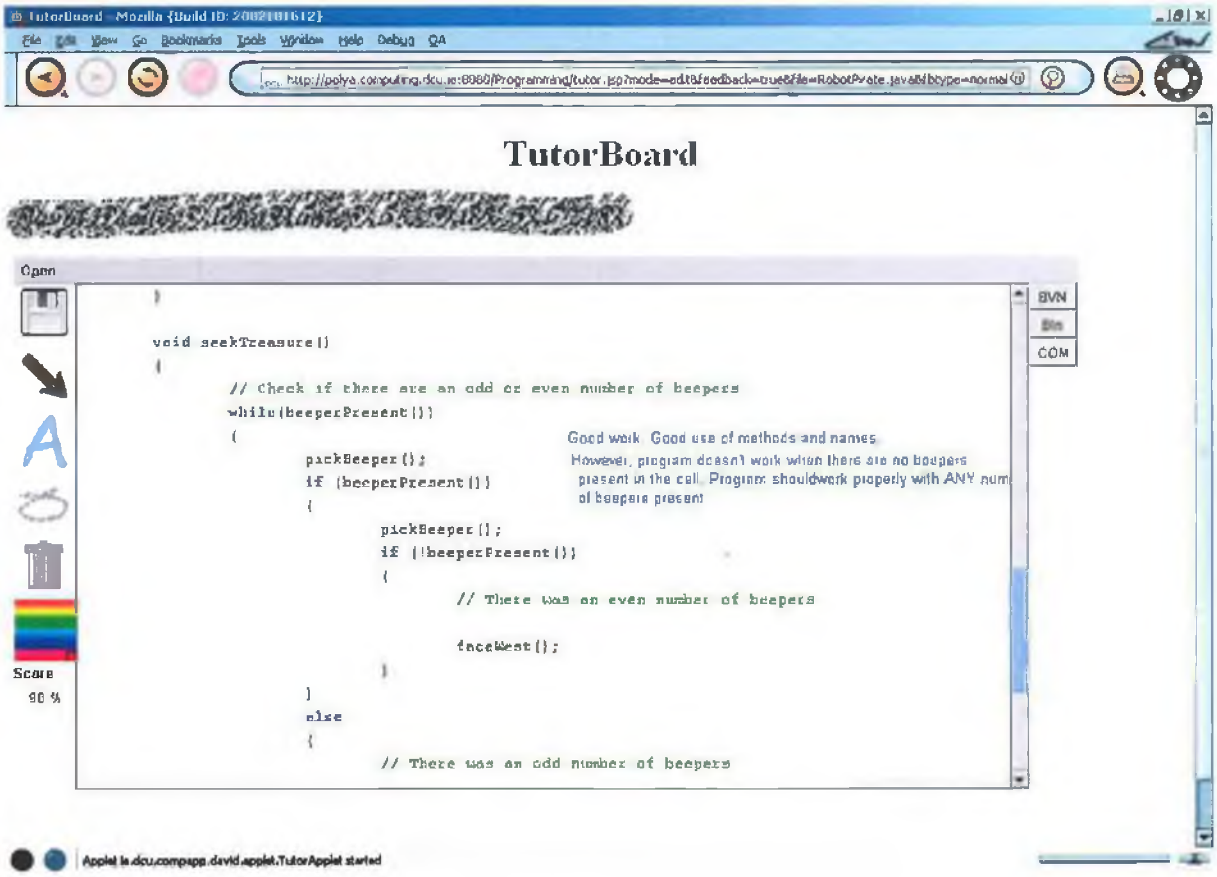
	Strongly Agree			Strongly Disagree	
I learnt problem solving skills from tutoring					
My java programming ability improved from tutoring					
Teaching the students reminded me of concepts and ideas from last year that I had forgotten about					

PART IV (ANY GENERAL COMMENTS)

- Please feel free to leave any general comments about your tutoring experience and the TutorBoard system.

THANK YOU !

Appendix D



TutorBoard - Mozilla {Build ID: 2002101612}
File Edit View Go Bookmarks Tools Window Help Debug QA
Open
SVN
Bin
COM
Score
90 %

http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=RobotPirate.java&btype=normal

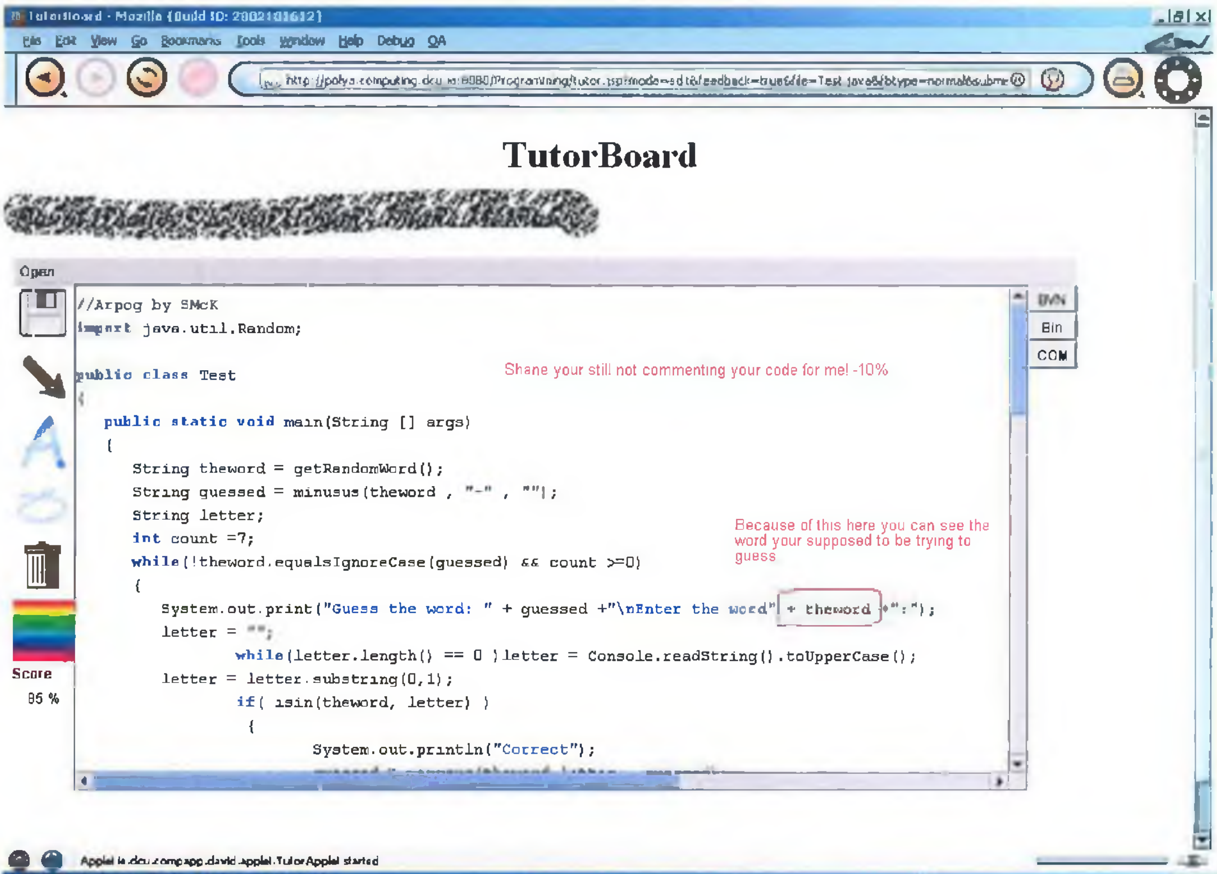
TutorBoard

[redacted]

```
}

void seekTreasure()
{
        // Check if there are an odd or even number of beepers
        while(beeperPresent())
        {
                pickBeeper();
                if (beeperPresent())
                {
                        pickBeeper();
                        if (!beeperPresent())
                        {
                                // There was an even number of beepers

                                faceWest();
                        }
                }
                else
                {
                        // There was an odd number of beepers
```

Good work. Good use of methods and names.
However, program doesn't work when there are no beepers present in the cell. Program should work properly with ANY num of beepers present

Applet ie.dcu.compapp.david.applet.TutorApplet started


TutorBoard - Mozilla {Build ID: 2002101612}

http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=Test.java&btype=normal&submit

TutorBoard

[redacted]

Open

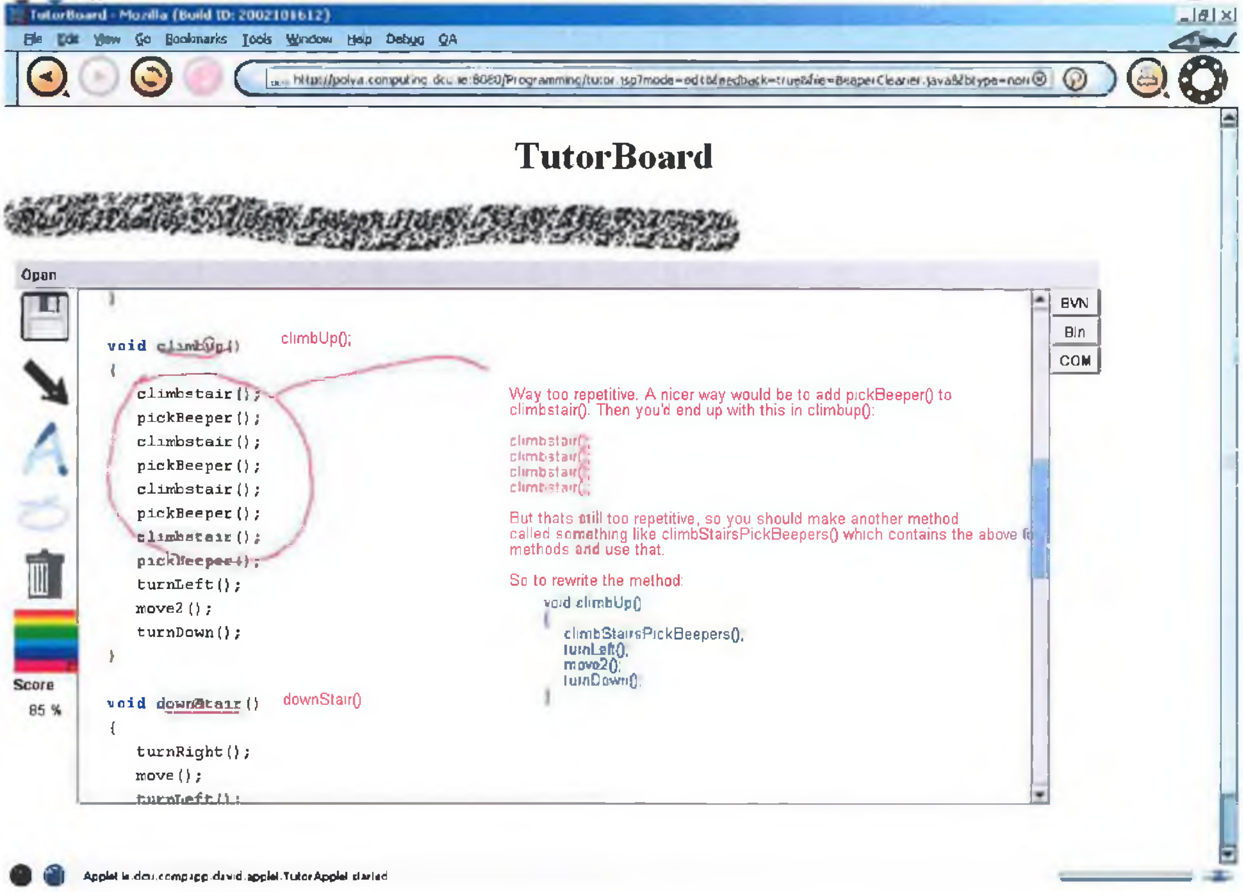
```
//Arpog by SMcK
import java.util.Random;

public class Test
{
   public static void main(String [] args)
   {
      String theword = getRandomWord();
      String guessed = minusus(theword , "-" , "");
      String letter;
      int count =7;
      while(!theword.equalsIgnoreCase(guessed) && count >=0)
      {
         System.out.print("Guess the word: " + guessed +"\nEnter the word" + theword +":");
         letter = "";
                 while(letter.length() == 0 )letter = Console.readString().toUpperCase();
         letter = letter.substring(0,1);
                 if( isin(theword, letter) )
                  {
                         System.out.println("Correct");
```

Shane your still not commenting your code for me! -10%

Because of this here you can see the word your supposed to be trying to guess

Score
85 %

Applet ie.dcu.compapp.david.applet.TutorApplet started


TutorBoard - Mozilla {Build ID: 2002101612}

http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=BeeperCleaner.java&btype=nor

TutorBoard

[redacted]

Open

```
 }

 void climbUp()
 {
    climbstair();
    pickBeeper();
    climbstair();
    pickBeeper();
    climbstair();
    pickBeeper();
    climbstair();
    pickBeeper();
    turnLeft();
    move2();
    turnDown();
 }

 void downStair()
 {
    turnRight();
    move();
    turnLeft();
```

climbUp();

Way too repetitive. A nicer way would be to add pickBeeper() to climbstair(). Then you'd end up with this in climbup():

climbstair();
climbstair();
climbstair();
climbstair();

But thats still too repetitive, so you should make another method called something like climbStairsPickBeepers() which contains the above methods and use that.

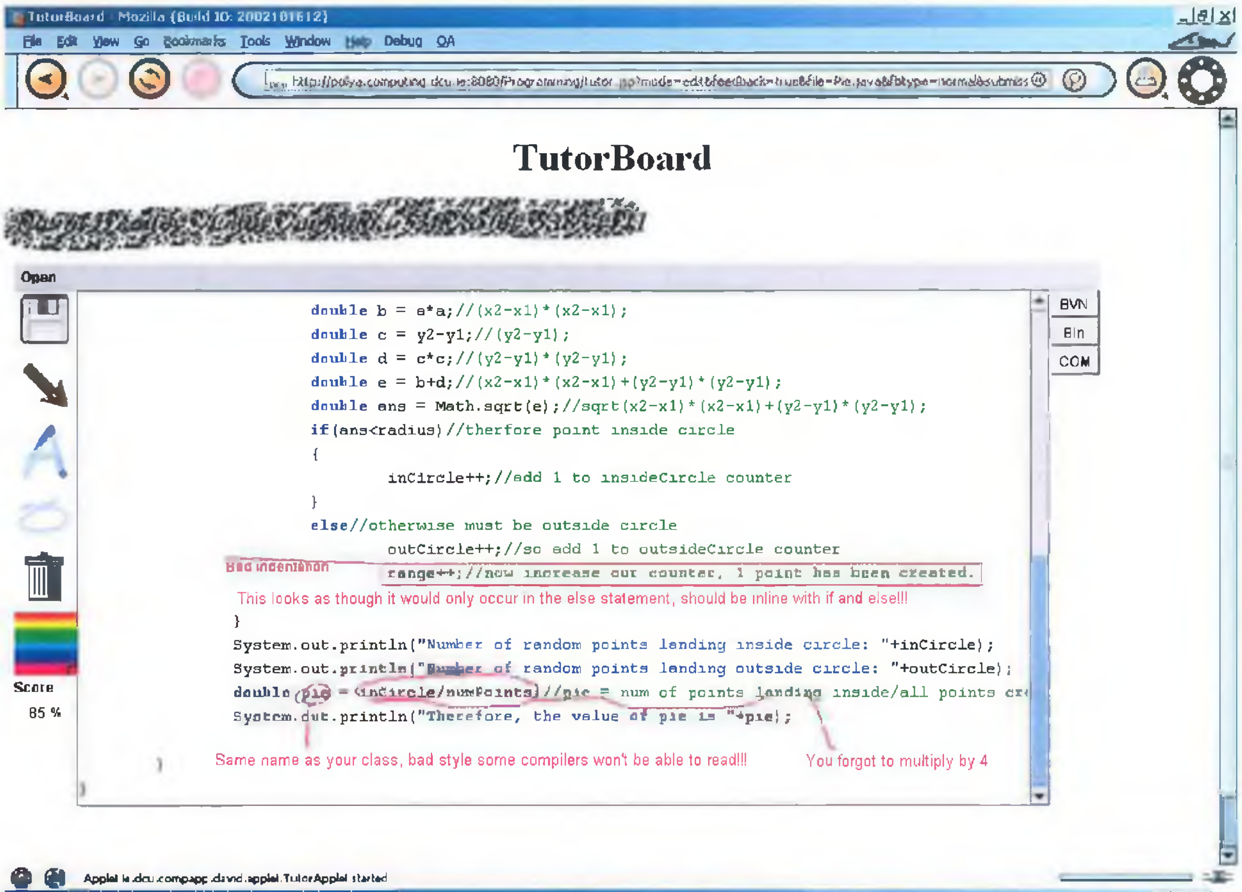
So to rewrite the method:

```
void climbUp()
{
  climbStairsPickBeepers();
  turnLeft();
  move2();
  turnDown();
}
```

downStair()

Score
85 %

Applet ie.dcu.compapp.david.applet.TutorApplet started


TutorBoard - Mozilla {Build ID: 2002101612}
File Edit View Go Bookmarks Tools Window Help Debug QA
http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=Pie.java&btype=normal&submis
Open
BVN
Bin
COM
Score
85 %
Applet ie.dcu.compapp.david.applet.TutorApplet started

TutorBoard

[redacted]

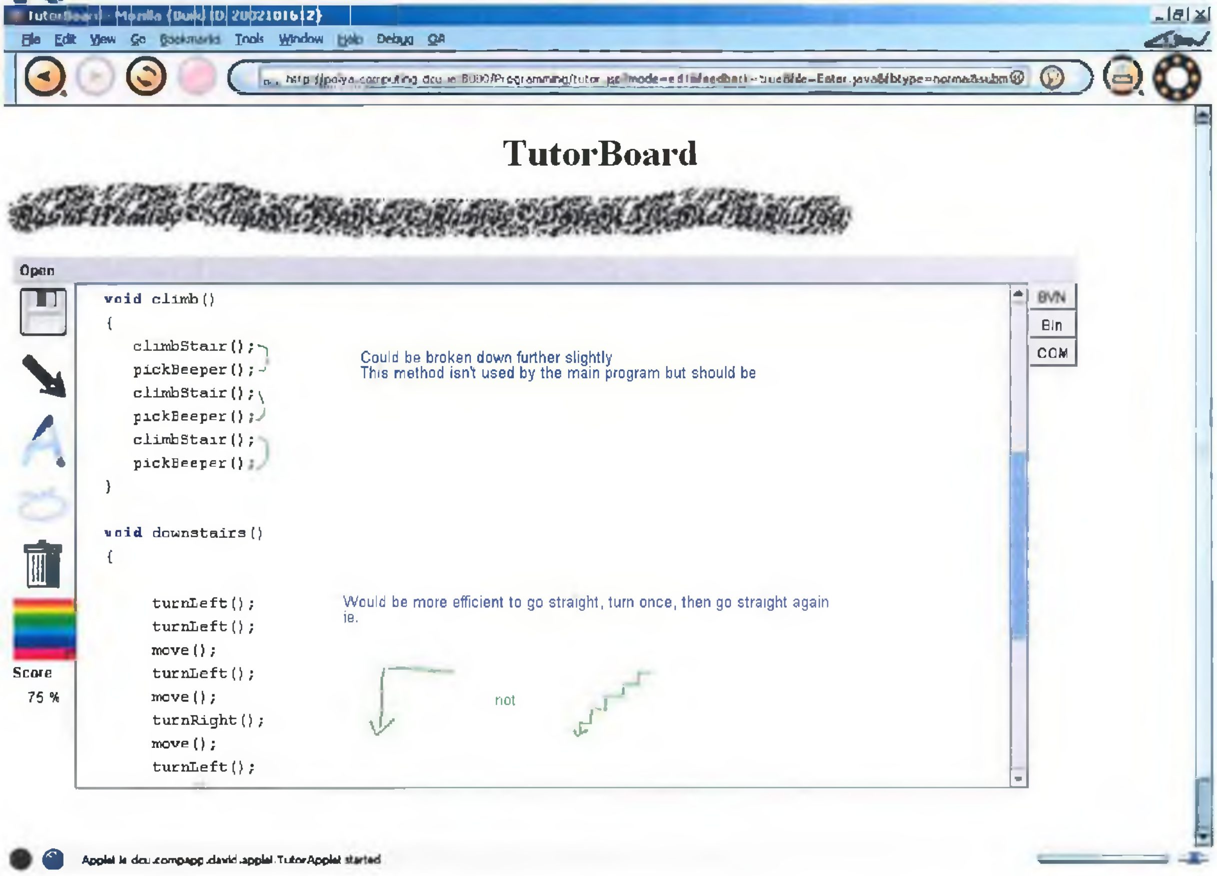
```
                    double b = a*a;//(x2-x1)*(x2-x1);
                    double c = y2-y1;//(y2-y1);
                    double d = c*c;//(y2-y1)*(y2-y1);
                    double e = b+d;//(x2-x1)*(x2-x1)+(y2-y1)*(y2-y1);
                    double ans = Math.sqrt(e);//sqrt(x2-x1)*(x2-x1)+(y2-y1)*(y2-y1);
                    if(ans<radius)//therfore point inside circle
                    {
                            inCircle++;//add 1 to insideCircle counter
                    }
                    else//otherwise must be outside circle
                            outCircle++;//so add 1 to outsideCircle counter
                            range++;//now increase our counter, 1 point has been created.
                }
                System.out.println("Number of random points landing inside circle: "+inCircle);
                System.out.println("Number of random points landing outside circle: "+outCircle);
                double pie = (inCircle/numPoints)//pie = num of points landing inside/all points cr
                System.out.println("Therefore, the value of pie is "+pie);

        }
}
```

Bad indentation

This looks as though it would only occur in the else statement, should be inline with if and else!!!

Same name as your class, bad style some compilers won't be able to read!!!

You forgot to multiply by 4


TutorBoard - Mozilla {Build ID: 2002101612}
File Edit View Go Bookmarks Tools Window Help Debug QA
http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=Enter.java&btype=normal&subm
Open
BVN
Bin
COM
Score
75 %
not
Applet ie.dcu.compapp.david.applet.TutorApplet started

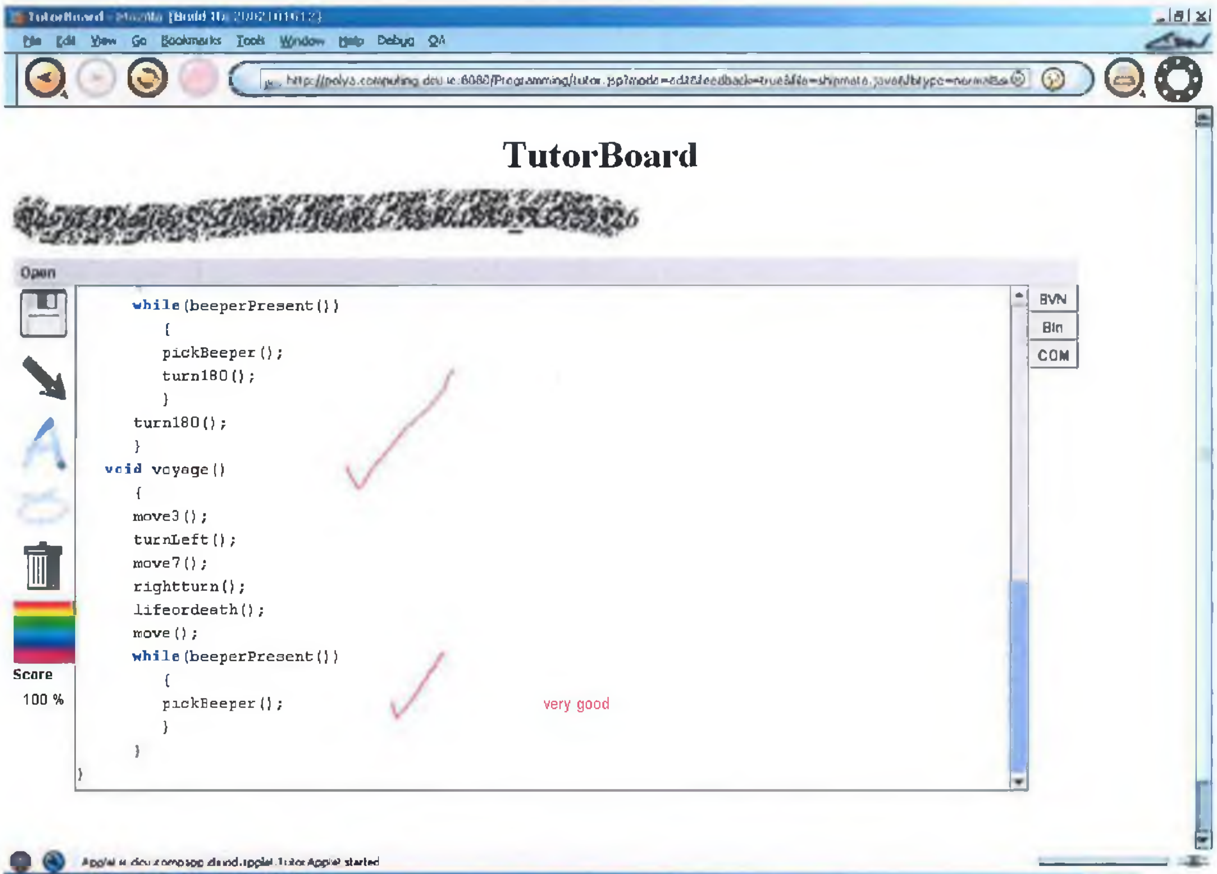
TutorBoard

[redacted]

```
   void climb()
   {
      climbStair();
      pickBeeper();
      climbStair();
      pickBeeper();
      climbStair();
      pickBeeper();
   }

   void downstairs()
   {

        turnLeft();
        turnLeft();
        move();
        turnLeft();
        move();
        turnRight();
        move();
        turnLeft();
```

Could be broken down further slightly
This method isn't used by the main program but should be

Would be more efficient to go straight, turn once, then go straight again
ie.


TutorBoard - Mozilla {Build ID: 2002101612}
File Edit View Go Bookmarks Tools Window Help Debug QA
http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=shipmate.java&btype=normal
Open
BVN
Bin
COM
Score

TutorBoard

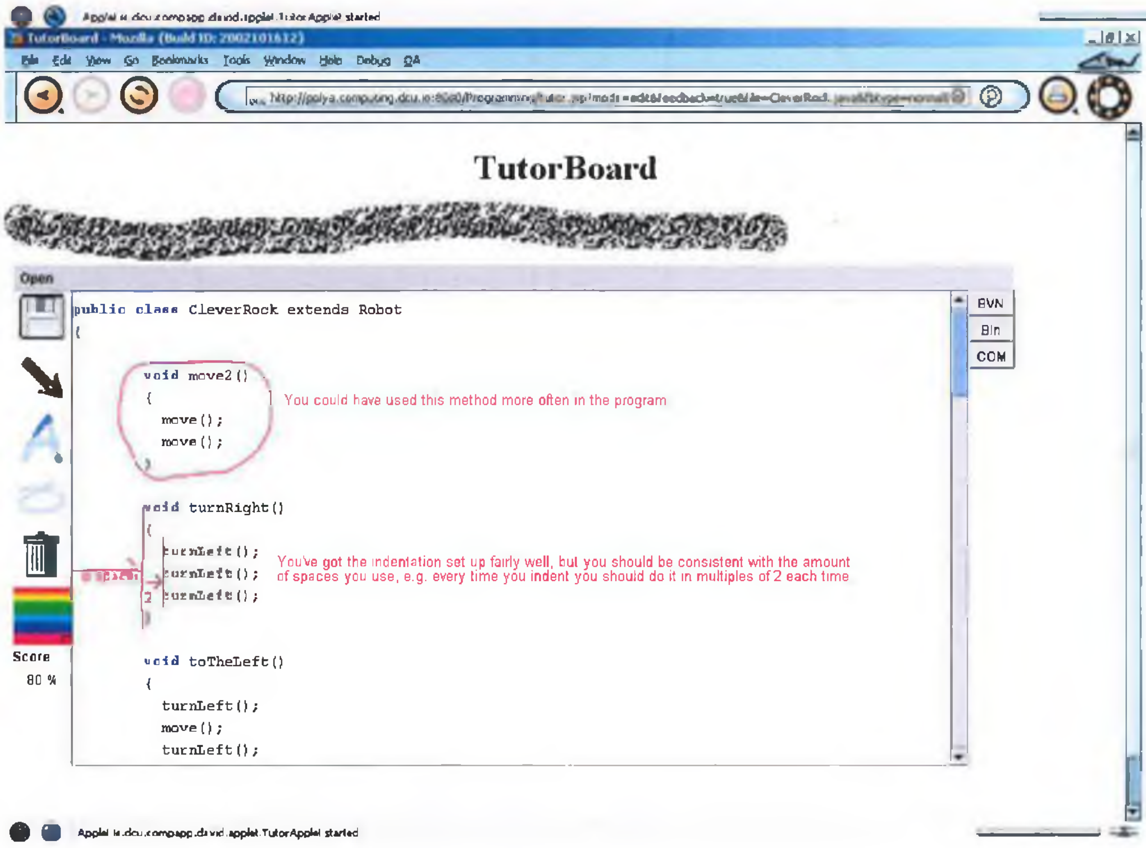
[redacted]

```
      while(beeperPresent())
         {
         pickBeeper();
         turn180();
         }
      turn180();
      }
   void voyage()
      {
      move3();
      turnLeft();
      move7();
      rightturn();
      lifeordeath();
      move();
      while(beeperPresent())
         {
         pickBeeper();
         }
      }
}
```

very good

Score 100 %

Applet ie.dcu.compapp.david.applet.TutorApplet started


TutorBoard - Mozilla (Build ID: 2002101612)
File Edit View Go Bookmarks Tools Window Help Debug QA
http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=CleverRock.java&btype=normal
Open
BVN
Bin
COM
Score

TutorBoard

[redacted]

```
public class CleverRock extends Robot
{

       void move2()
       {
         move();
         move();
       }

       void turnRight()
       {
         turnLeft();
         turnLeft();
         turnLeft();
       }

       void toTheLeft()
       {
         turnLeft();
         move();
         turnLeft();
```

You could have used this method more often in the program

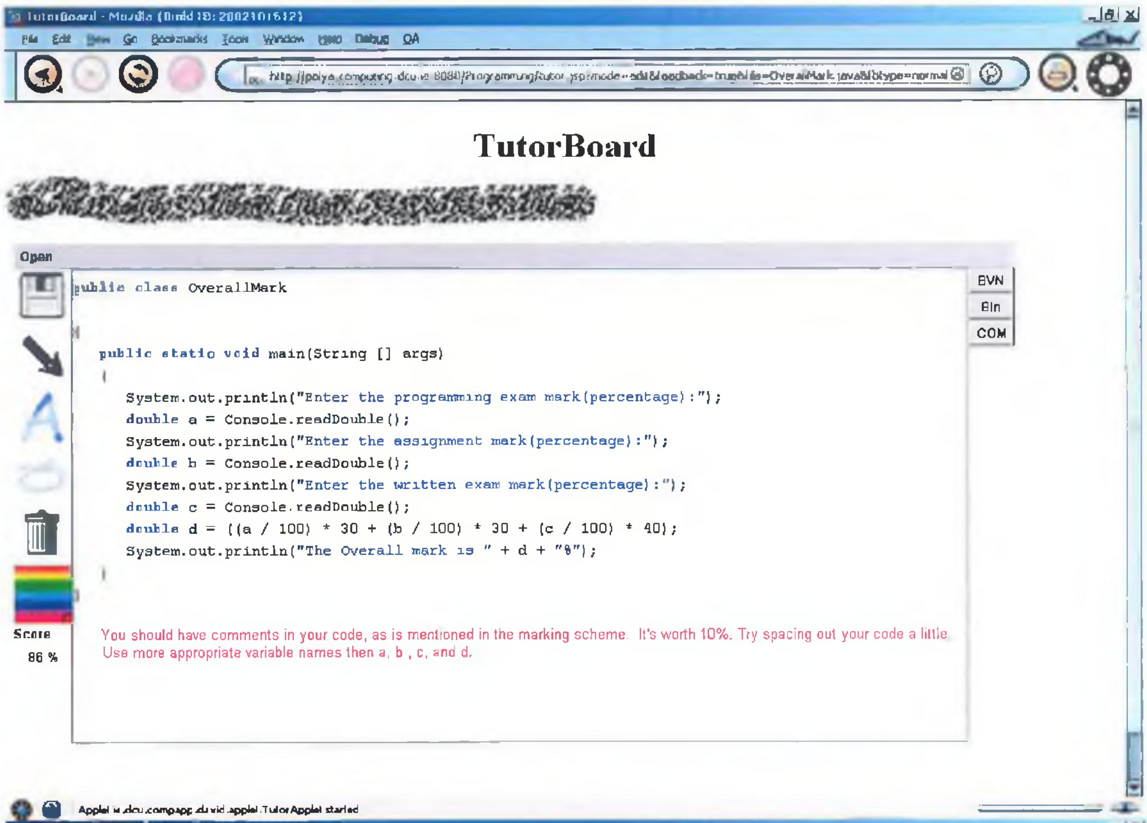
spaces

2

You've got the indentation set up fairly well, but you should be consistent with the amount of spaces you use, e.g. every time you indent you should do it in multiples of 2 each time

Score 80 %

Applet ie.dcu.compapp.david.applet.TutorApplet started


TutorBoard - Mozilla (Build ID: 2002101612)

http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=OverallMark.java&btype=normal

TutorBoard

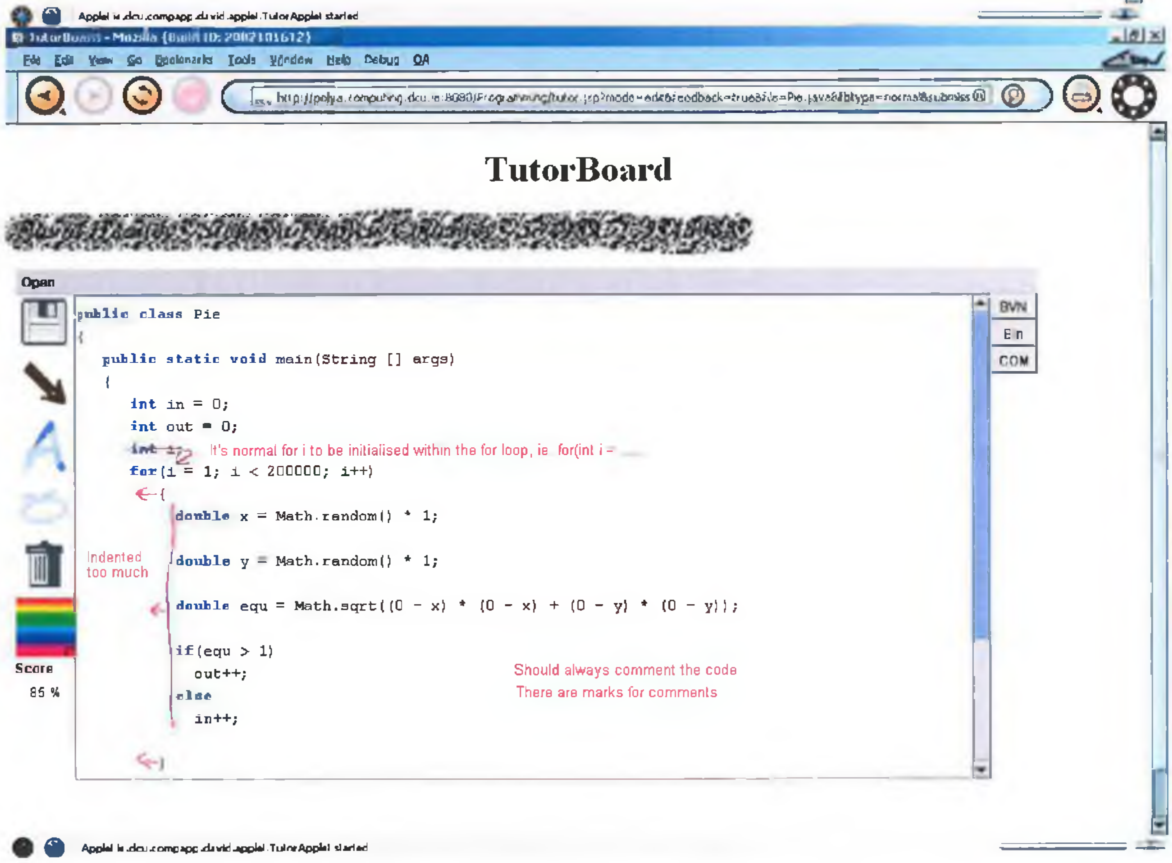
[redacted]

```
public class OverallMark
{
  public static void main(String [] args)
  {
    System.out.println("Enter the programming exam mark(percentage):");
    double a = Console.readDouble();
    System.out.println("Enter the assignment mark(percentage):");
    double b = Console.readDouble();
    System.out.println("Enter the written exam mark(percentage):");
    double c = Console.readDouble();
    double d = ((a / 100) * 30 + (b / 100) * 30 + (c / 100) * 40);
    System.out.println("The Overall mark is " + d + "%");
  }
```

You should have comments in your code, as is mentioned in the marking scheme. It's worth 10%. Try spacing out your code a little. Use more appropriate variable names then a, b, c, and d.

Score
86 %

Applet ie.dcu.compapp.david.applet.TutorApplet started


TutorBoard - Mozilla (Build ID: 2002101612)

http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=Pie.java&btype=normal&submiss

TutorBoard

[redacted]

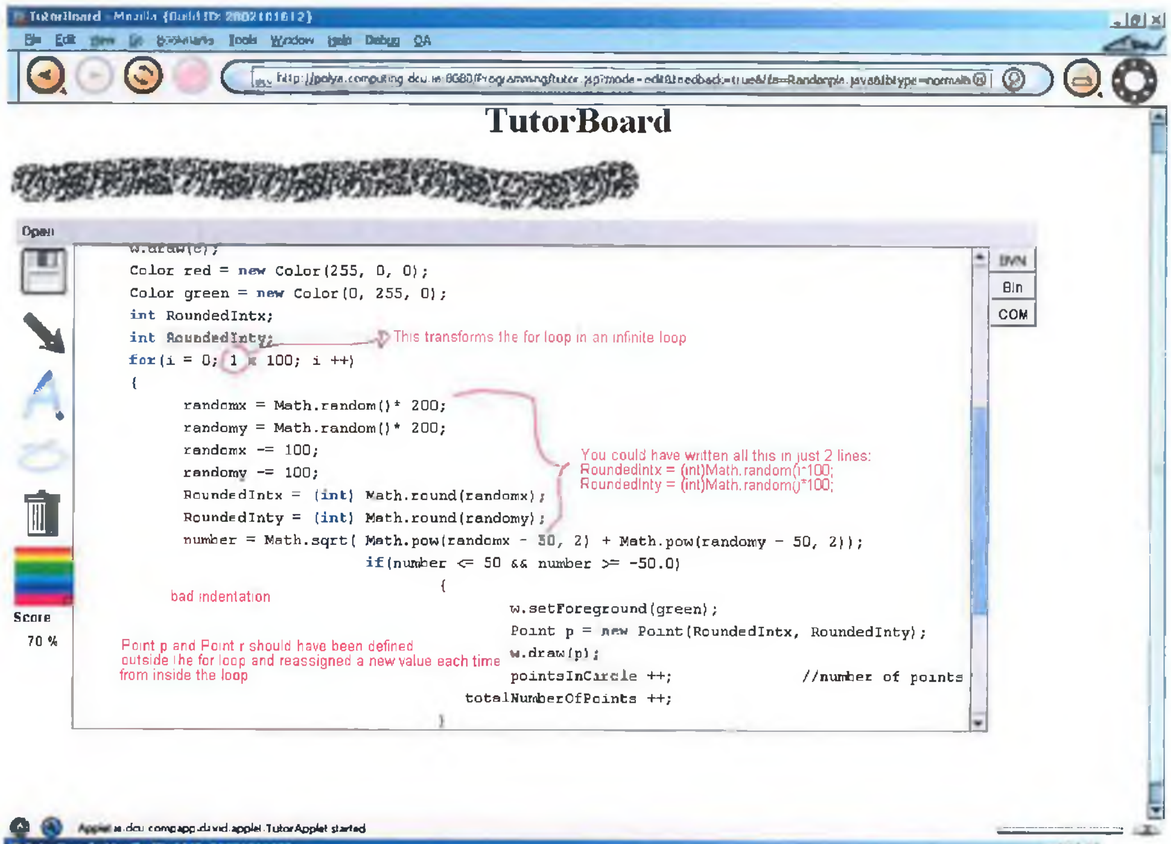
```
public class Pie
{
  public static void main(String [] args)
  {
    int in = 0;
    int out = 0;
    ~~int i;~~ 2   It's normal for i to be initialised within the for loop, ie. for(int i = ...
    for(i = 1; i < 200000; i++)
    {
      double x = Math.random() * 1;

      double y = Math.random() * 1;

      double equ = Math.sqrt((0 - x) * (0 - x) + (0 - y) * (0 - y));

      if(equ > 1)
        out++;
      else
        in++;

    }
```

Indented too much

Should always comment the code
There are marks for comments

Score
85 %

Applet ie.dcu.compapp.david.applet.TutorApplet started


TutorBoard - Mozilla {Build ID: 2002101612}
File Edit View Go Bookmarks Tools Window Help Debug QA
http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=Randompi.java&btype=normal
Open
BVN
Bin
COM
Score
70 %
Applet ie.dcu.compapp.david.applet.TutorApplet started

TutorBoard

[redacted]

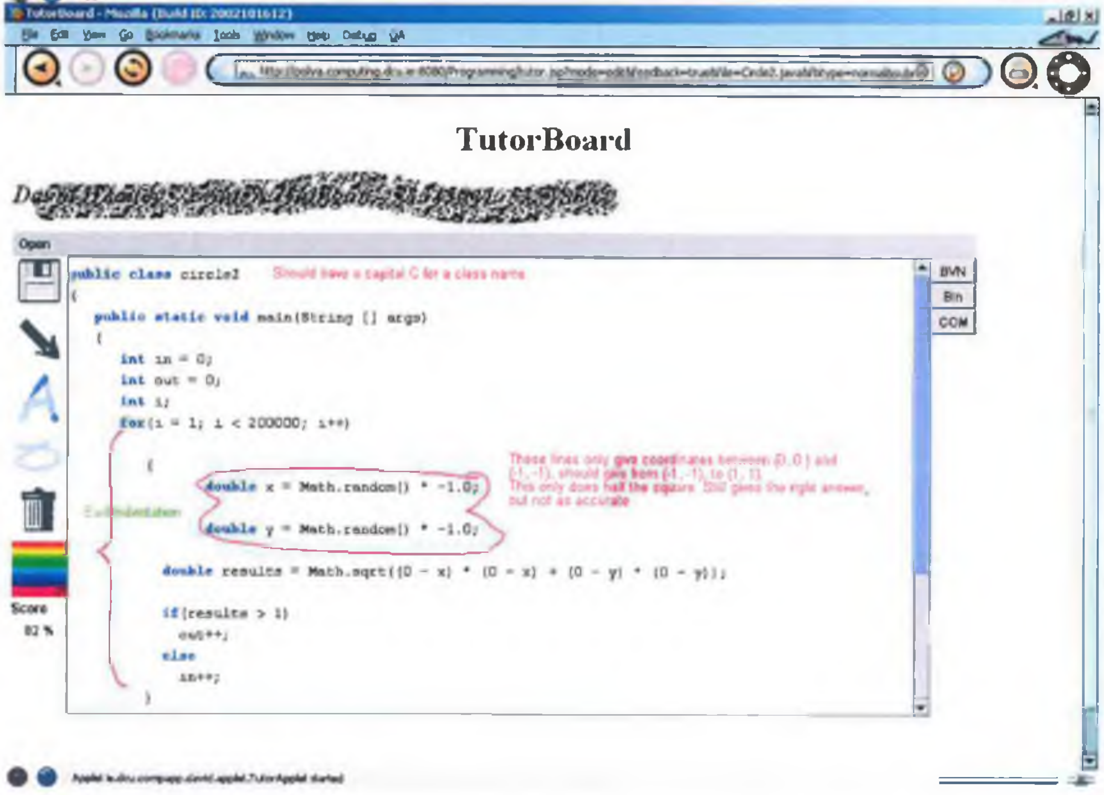
```
        w.draw(c);
        Color red = new Color(255, 0, 0);
        Color green = new Color(0, 255, 0);
        int RoundedIntx;
        int RoundedInty;
        for(i = 0; 1 < 100; i ++)
        {
              randomx = Math.random()* 200;
              randomy = Math.random()* 200;
              randomx -= 100;
              randomy -= 100;
              RoundedIntx = (int) Math.round(randomx);
              RoundedInty = (int) Math.round(randomy);
              number = Math.sqrt( Math.pow(randomx - 50, 2) + Math.pow(randomy - 50, 2));
                                   if(number <= 50 && number >= -50.0)
                                            {
                                                   w.setForeground(green);
                                                   Point p = new Point(RoundedIntx, RoundedInty);
                                                   w.draw(p);
                                                   pointsInCircle ++;              //number of points
                                             totalNumberOfPoints ++;
                                            }
```

This transforms the for loop in an infinite loop

You could have written all this in just 2 lines:
RoundedIntx = (int)Math.random()*100;
RoundedInty = (int)Math.random()*100;

bad indentation

Point p and Point r should have been defined outside the for loop and reassigned a new value each time from inside the loop


TutorBoard - Mozilla (Build ID: 2002101612)
File Edit View Go Bookmarks Tools Window Help Debug QA
http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=Circle2.java&btype=normal
Open
BVN
Bin
COM
Score
82 %
Applet ie.dcu.compapp.david.applet.TutorApplet started

TutorBoard

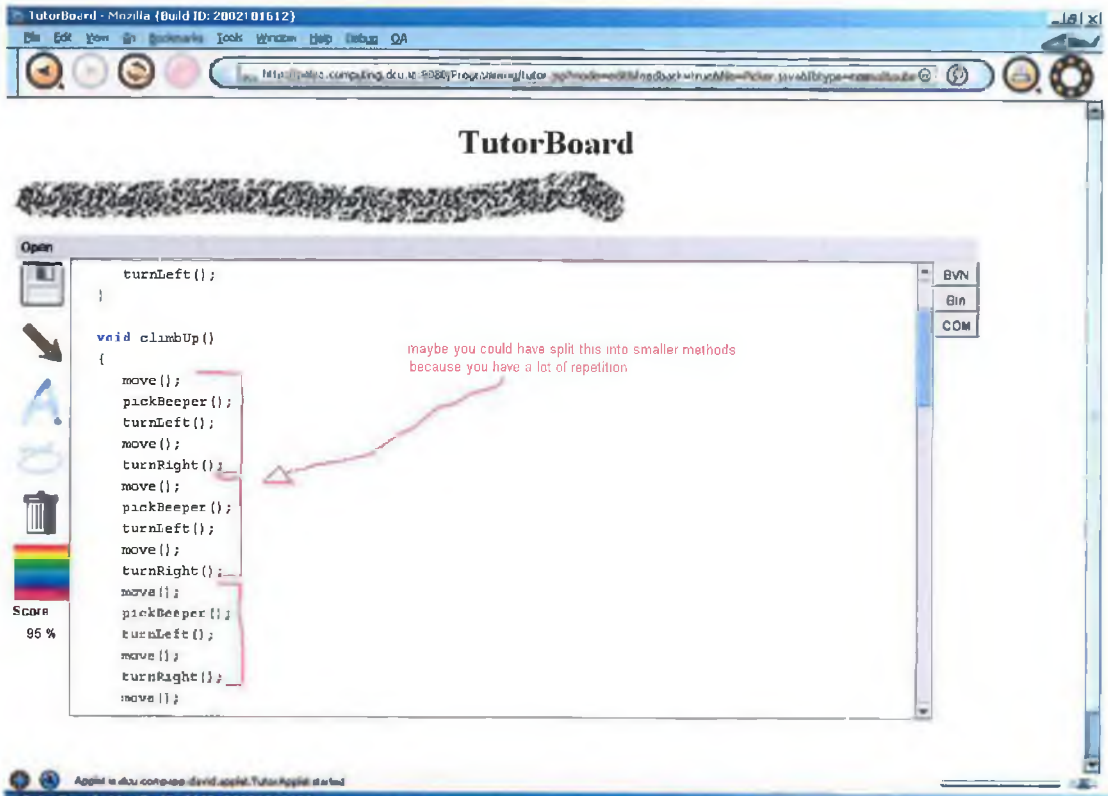
[redacted]

```
public class circle2
{
   public static void main(String [] args)
   {
      int in = 0;
      int out = 0;
      int i;
      for(i = 1; i < 200000; i++)

         {
               double x = Math.random() * -1.0;

               double y = Math.random() * -1.0;

         double results = Math.sqrt((0 - x) * (0 - x) + (0 - y) * (0 - y));

         if(results > 1)
           out++;
         else
           in++;
         }
```

Should have a capital C for a class name

These lines only give coordinates between (0,0) and (-1,-1), should give from (-1,-1) to (1,1). This only does half the square. Still gives the right answer, but not as accurate

[illegible]

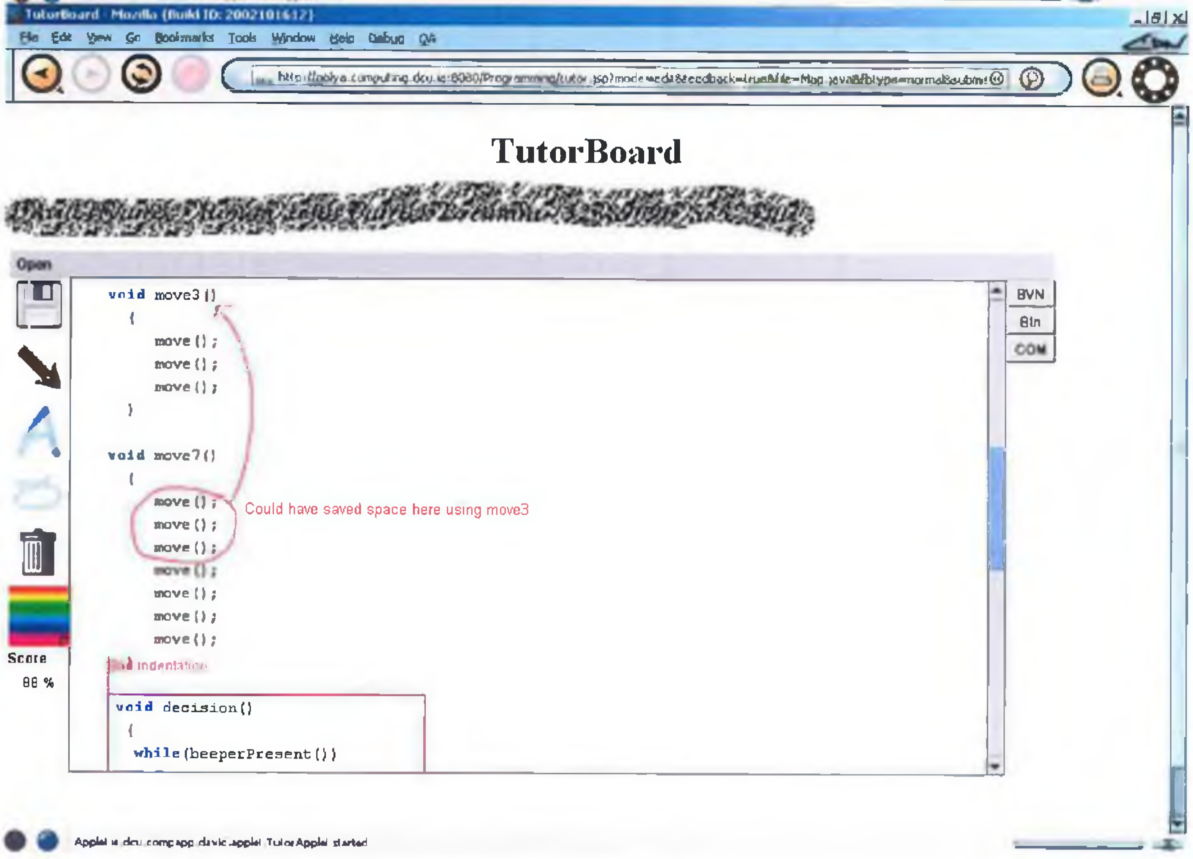

TutorBoard - Mozilla {Build ID: 2002101612}
File Edit View Go Bookmarks Tools Window Help Debug QA
Open
BVN
Bin
COM
Score
95 %
Applet ie.dcu.compapp.david.applet.TutorApplet started

TutorBoard

[redacted]

```
    turnLeft();
  }

void climbUp()
{
    move();
    pickBeeper();
    turnLeft();
    move();
    turnRight();
    move();
    pickBeeper();
    turnLeft();
    move();
    turnRight();
    move();
    pickBeeper();
    turnLeft();
    move();
    turnRight();
    move();
```

maybe you could have split this into smaller methods
because you have a lot of repetition


TutorBoard - Mozilla {Build ID: 2002101612}
File Edit View Go Bookmarks Tools Window Help Debug QA
Open
BVN
Bin
COM
Score
88 %
Applet ie.dcu.compapp.david.applet.TutorApplet started

TutorBoard

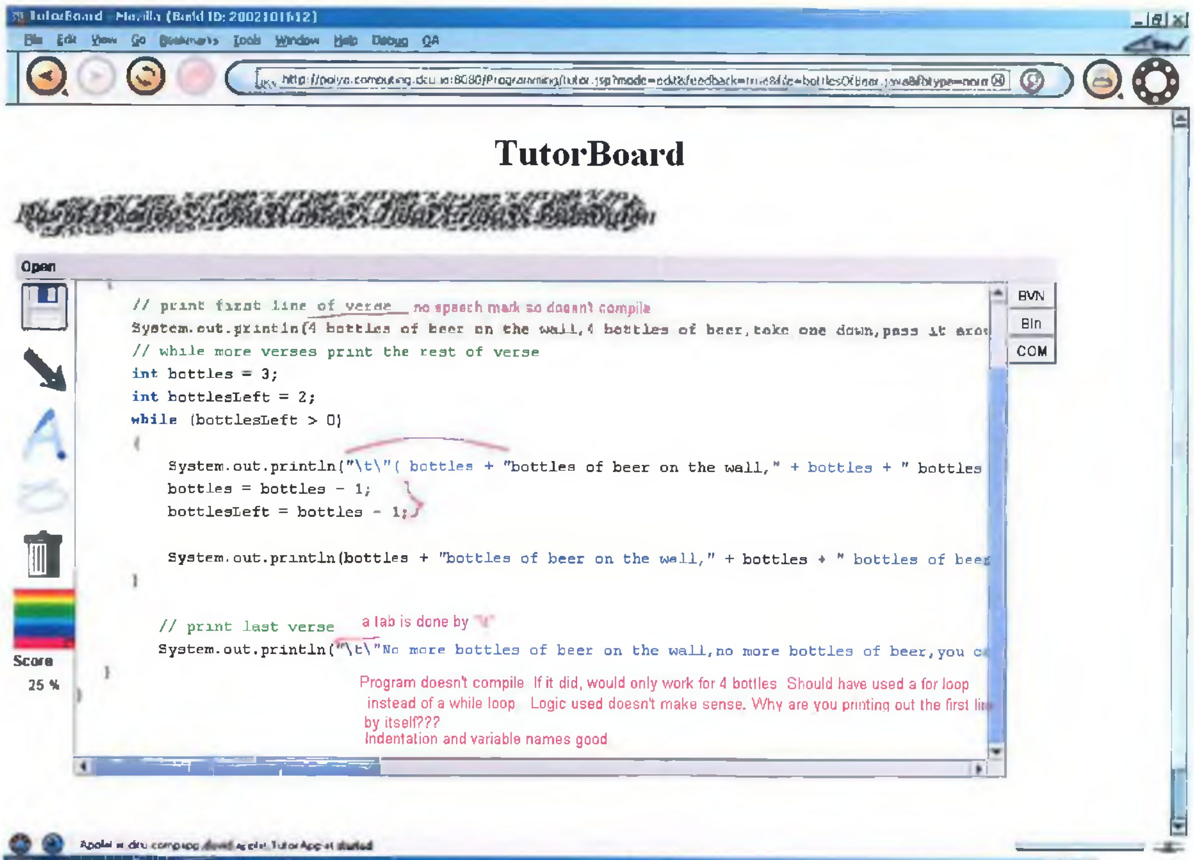
[redacted]

```
void move3()
  {
      move();
      move();
      move();
  }

void move7()
  {
      move();
      move();
      move();
      move();
      move();
      move();
      move();
```

Could have saved space here using move3

Bad indentation

```
void decision()
 {
  while(beeperPresent())
```


TutorBoard - Mozilla (Build ID: 2002101612)
File Edit View Go Bookmarks Tools Window Help Debug QA
http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=bottlesOfBeer.java&fbtype=norm
Open
BVN
Bln
COM
Applet ie.dcu.compapp.david.applet.TutorApplet started

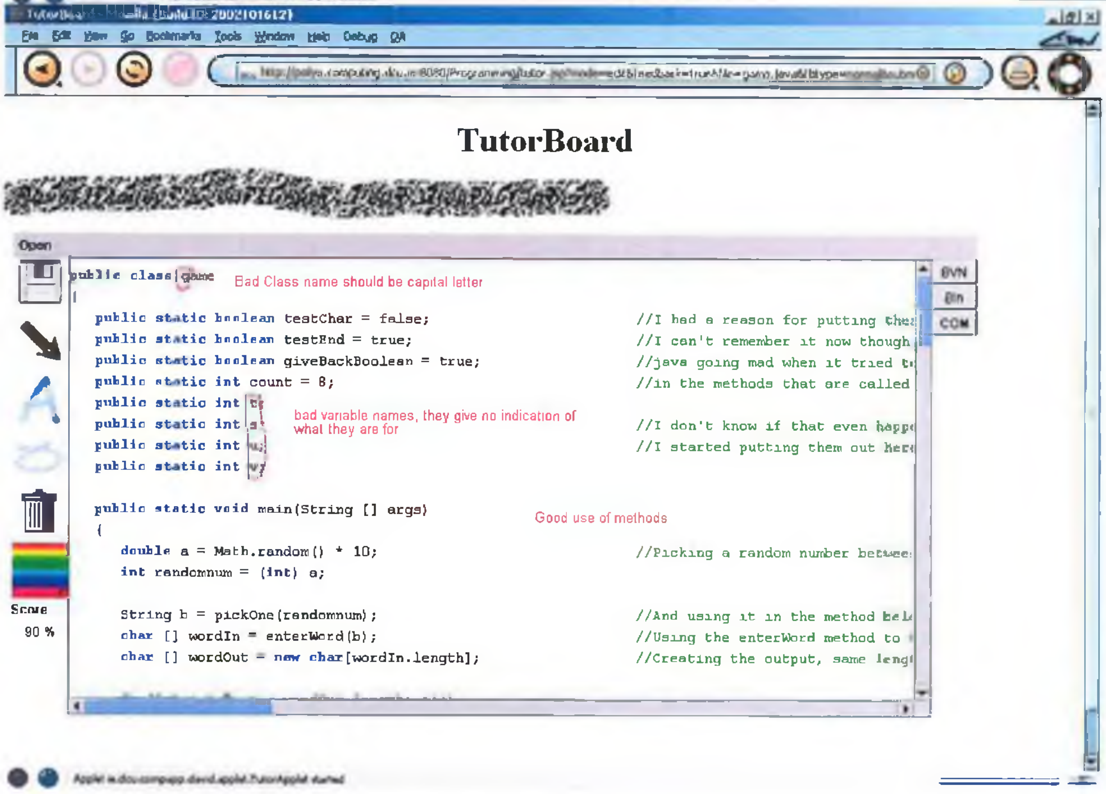
TutorBoard

[redacted]

```
   // print first line of verse   no speech mark so doesn't compile
   System.out.println(4 bottles of beer on the wall,4 bottles of beer,take one down,pass it aro
   // while more verses print the rest of verse
   int bottles = 3;
   int bottlesLeft = 2;
   while (bottlesLeft > 0)
   {
       System.out.println("\t\"( bottles + "bottles of beer on the wall," + bottles + " bottles
       bottles = bottles - 1;
       bottlesLeft = bottles - 1;

       System.out.println(bottles + "bottles of beer on the wall," + bottles + " bottles of beer
   }

     // print last verse    a lab is done by "\t"
     System.out.println("\t\"No more bottles of beer on the wall,no more bottles of beer,you c
 }
}
```

Program doesn't compile If it did, would only work for 4 bottles Should have used a for loop instead of a while loop Logic used doesn't make sense. Why are you printing out the first li by itself???
Indentation and variable names good

Score
25 %


TutorBoard - Mozilla (Build ID: 2002101612)
File Edit View Go Bookmarks Tools Window Help Debug QA
http://polya.computing.dcu.ie:8080/Programming/tutor.jsp?mode=edit&feedback=true&file=game.java&fbtype=normal
Open
BVN
Bln
COM
Applet ie.dcu.compapp.david.applet.TutorApplet started

TutorBoard

[redacted]

```
public class game   Bad Class name should be capital letter
{
  public static boolean testChar = false;                     //I had a reason for putting the
  public static boolean testEnd = true;                       //I can't remember it now though
  public static boolean giveBackBoolean = true;               //java going mad when it tried t
  public static int count = 8;                                //in the methods that are called
  public static int t;
  public static int s;    bad variable names, they give no indication of   //I don't know if that even happ
  public static int u;    what they are for                                //I started putting them out her
  public static int v;

  public static void main(String [] args)          Good use of methods
  {
     double a = Math.random() * 10;                           //Picking a random number betwee
     int randomnum = (int) a;

     String b = pickOne(randomnum);                           //And using it in the method bel
     char [] wordIn = enterWord(b);                           //Using the enterWord method to
     char [] wordOut = new char[wordIn.length];               //Creating the output, same leng
```

Score
90 %