Mining Autograding Data in Computer Science Education

Vincent Gramoli
NICTA and
University of Sydney
NSW, 2006
Australia
vincent.gramoli@sydney.edu.au

Michael Charleston
University of Tasmania
Sandy Bay, TAS 7005
University of Sydney
NSW, 2006
Australia
michael.charleston@utas.edu.au

Bryn Jeffries
University of Sydney
NSW, 2006
Australia
bryn.jeffries@sydney.edu.au

Irena Koprinska
University of Sydney
NSW, 2006
Australia
irena.koprinska@sydney.edu.au

Martin McGrane
University of Sydney
NSW, 2006
Australia
mmcg5982@uni.sydney.edu.au

Alex Radu
University of Sydney
NSW, 2006
Australia
arad0726@uni.sydney.edu.au

Anastasios Viglas
University of Sydney
NSW, 2006
Australia
taso.viglas@sydney.edu.au

Kalina Yacef
University of Sydney
NSW, 2006
Australia
kalina.yacef@sydney.edu.au

ABSTRACT

In this paper we present an analysis of the impact of instant feedback and autograding in computer science education, beyond the classic Introduction to Programming subject.

We analysed the behaviour of 1st year to 4th year students when submitting programming assignments at the University of Sydney over a period of 3 years. These assignments were written in different programming languages, such as C, C++, Java and Python, for diverse computer science courses, from fundamental ones—algorithms, complexity, formal languages, data structures and artificial intelligence to more "practical" ones—programming, distributed systems, databases and networks.

We observed that instant feedback and autograding can help students and instructors in subjects not necessarily focused on programming. We also discuss the relationship between the student performance in these subjects and the choice of programming languages or the times at which a student starts and stops working on an assignment.

Categories and Subject Descriptors

K.3 [**Computer and Information Science Education**]: Computer science education; H.2 [**Database Applications**]: Data mining

General Terms

Measurement, Human Factors

Keywords

Educational data mining, instant feedback, learning analytics.

1. INTRODUCTION

Autograding and instant feedback are known to be effective in helping both the instructors of programming subjects, by automating basic tests in programming assignments to help marking, and their students, by providing them with instant feedback that helps them rapidly correcting their programming mistakes. For example, different cohorts of students enrolled in an introductory C programming subject submitted on average twice as many times when they were receiving instant feedback on their submitted work [16]. With the advent of the online courses, both for massive open (MOOC) and smaller and private (SPOC) audiences [8], these techniques will certainly gain in popularity to automate courses a step further. Until now, however, autograding and instant feedback solutions for programming assignments have been mainly focused on subjects exclusively focused on programming skills.

Applying autograding beyond the traditional "Introduction to Programming" courses raises new challenges. First, developing basic programming skills becomes less of a goal, and instead these programming skills are assumed as a prerequisite. Second, students are also expected to understand the implications of their program, for example, its complexity, its performance and its interoperability with other programs, sometimes regardless of the program syntax. Third, general computer science courses require the student to have passed a programming course, regardless of her true performance, but do not aim at providing instant feedback regarding the quality of the code produced. Fourth, while it is

Course code	Name	Level	#Semesters	#Students
Info1103	Introduction to Programming	1st year	6	1815
Info1105	Data Structures	1st year	5	1104
Info1905	Advanced Data Structures	1st year	5	89
Comp2007	Algorithms and Complexity	2nd year	1	194
Comp2907	Algorithms and Complexity (Advanced)	2nd year	1	22
Comp2022	Formal Languages and Logic	2nd year	1	84
Comp2121	Distributed Systems and Network Principles	2nd year	2	96
Comp3308	Artificial Intelligence	3rd year	1	121
Comp3608	Artificial Intelligence (Advanced)	3rd year	1	25
Info3404	Database Systems 2	3rd year	1	90
Info3504	Advanced Database Systems 2	3rd year	1	9
Comp5416	Advanced Networks	4th year	1	27
Comp5211	Algorithms	4th year	1	26
Total	13 subjects	1st-4th year	27 semesters	3702

Table 1: Courses, levels, semester count and student count of the analysed dataset

typically the case for "Introduction to Programming" courses to prescribe a specific programming language, other courses give the student the flexibility to work in a language of their choice.

We designed an instant feedback and autograding tool named Pasta. Pasta helps instructors assessing the skills of students and provides these students with instant feedback not exclusively based on their programming skills but also on their ability to design algorithms, understand network protocols, test databases, reason logically, measure complexity, etc. We extracted the data from 1st to 4th year students stored by the Pasta tool over the years of 2013, 2014 and 2015 in 13 computer science subjects, each running for at least one semester at the University of Sydney, as depicted in Table 1. We focused on data related to the submission times, the programming language used, the efforts in coding, the weight in the final mark and the due date of assignments, and the overall performance of each student. At the end of this study, we asked some students about their experience on the use of our autograding and instant feedback tool, Pasta.

In this paper, we demonstrate that autograding and instant feedback can be applied to the computer science curriculum in general. In particular, we mined PASTA submission data in various subjects ranging from fundamental ones like algorithms, complexity, logic, data structures and artificial intelligence, to more applied subjects, like programming, distributed systems, databases and networks. The results of this study show that (i) instant feedback and autograding are praised by students and affect the way they submit assignments, and help instructors especially in subjects not focused on assessing programming skills; that (ii) student results vary significantly depending on the programming language they choose to code a programming task; and that (iii) the earlier students start submitting an assignment, the earlier the students feel the assignment to be ready, presumably based on the instant feedback corresponding to these submissions.

The paper is organized as follows. In Section 2 we discuss related work. In Section 3 we present our autograding and instant feedback tool, Pasta. In Section 4 we motivate the need for instant feedback in computer science. In Section 5 we study the efforts students spent while accessing the system and the performance they attained. In Section 6 we analyse the choice of programming languages and the relation with student performance. In Section 7 we list differences in the way different instructors used Pasta to assess different skills. In Section 8 we study the feedback obtained from a sample of students regarding the use of Pasta during previous years. Finally, we conclude in Section 10.

2. RELATED WORK

The use of autograding systems in introductory computer programming courses has been reported many times [1,3,5, 16,18–20]. Our work extends previous research by studying the impact of autograding and instant feedback in not only introductory computer programming courses but also in the broader context of computer science education, where the focus is on using computer programming as a tool to solve subject specific problems.

An example of an autograding system used in a first year programming course is the *submit* system [20], developed at the Victoria University of Technology in Melbourne and used to teach Java. The paper reports on a number of initial trials with a relatively small number of students (less than 100). Feedback from both students and staff was very positive—students were able to submit multiple times, each time receiving instant and individual feedback, they would keep improving their program until it was accepted by *submit* without errors. The marking workload for staff was also significantly reduced, and staff could spend more time helping students with less mundane questions.

The autograding system Pasta and two other sources, the discussion board Piazza and the assessment marks, were used to detect the students at risk of failing in a first year programming course in Java [12]. A decision tree was built achieving 87% accuracy in predicting whether students will pass or fail their final exam, from data available in the middle of the semester. The results also showed that using information from the autograding system improves the accuracy, in comparison to only using the assessment marks. The decision tree rules indicated the importance of starting the assessments early and also finishing them early. In [11] data from the same sources was used to characterise and predict the performance of three groups of students, high-, average- and low-performing. Our dataset is bigger: it includes the course used in these two studies but it uses data for 9 consecutive offerings of this course instead of 1 semester only, and also uses data for 12 other courses.

Web-CAT is an automated grading system for C++ and Java developed at Virginia Tech [6]. Students are required to submit not only their code but also unit tests for their code, both of which are marked by the system. By writing tests for their own code students learn more as they need to demonstrate the correctness of their code, which results in a higher-quality code.

Kattis [7] is an automated assessment system developed at the Royal Institute of Technology, Sweden, being used to help recruiting talented programmers and also for various undergraduate courses, e.g., programming, algorithms, data structures, complexity and networks. For courses such as Advanced Algorithms, where the goal is not only to write correct but also efficient programs, Kattis checks time limits and assign scores based on how fast the solution is.

Bottlenose is a web-based autograding system that was used in a first year C programming course [16] without limiting the submission count per student. A comparison between the student behaviour on the same assignments when using and not using Bottlenose was conducted. The results showed that the number of submissions per student per assignment was significantly higher when using Bottlenose, which was attributed to students making use of the feedback to improve their programs.

A recent method for providing feedback to students in programming courses was presented in [18]. Starting from a reference implementation of the solution and an error model that includes potential corrections to errors that students might make, the system automatically derives correction rules that can be used to provide feedback to students about how incorrect their solution is and where the problems are. The system was tested in an introductory programming course using Python and showed promising results.

Autolab [14] is an autograding system developed at Carnegie Mellon University and used for a first year programming course in C. A distinct feature of Autograde is the real-time scoreboard that shows the class performance on the autograded assessment in ranked order. It was found to create a healthy competition encouraging students to improve their assignments and do them quicker.

CodeWrite [5] is a an automated system for the introductory Java programming course at the University of Auckland that uses a test driven learning pedagogy. It allows students to develop exercises by providing a problem description, reference implementation and a set of simple test cases. Once the student has written a fully consistent exercise, in which the reference implementation passes the accompanying tests, it can be made available to the student's peers.

3. INSTANT FEEDBACK SOFTWARE

At the University of Sydney, many computer science courses use an autograding and instant feedback tool developed in-house, named PASTA, standing for "Programming Assessment Submission and Testing Application". This tool has been used to help instructors assess the skills of students and to provide these students with instant feedback not exclusively based on their programming skills but also on other important skills such as their ability to design algorithms, understand network protocols, test databases, reason logically, understand complexity, etc.

3.1 Student usage

PASTA is an integrated program that provides a web interface for students to submit programs (written in C, C++, Python or Java) or Matlab code snippets, intended to solve a task or an assignment as defined by the instructor. PASTA is configured and accessed through a web interface, runs with Tomcat and centralises the information into dedicated folders and a database through the Hibernate library. Students upload their attempts as `zip` archives via the web interface. The archive is unzipped, processed, compiled and run against the unit tests automatically by the server(s). Note that the tests can be written in Java using the JUnit package even though the program to be tested may be written in a different programming language than Java.

In the current setup the submissions are tentatively compiled and, upon compilation success, are placed in a single queue. The web interface shows synchronously the compilation errors or updates asynchronously once testing has completed, returning a message with information about which tests passed and failed. Then some results are returned to the student in the form of a webpage indicating all the series of tests that were run against the submitted code, whether the tests were successful and the appropriate feedback for each test failed. Students can make use of this information to improve their work and possibly resubmit a better version.

3.2 Instructor usage

Through PASTA's admin interface, the instructor can set up the maximum submission count per student, upload (unit) tests and specify the level of immediate feedback to provide to the students. Some of the tests may be "hidden":, i.e they do not provide feedback to students upon submission. Such tests are used to prevent students from over-optimizing a solution for tests rather than implementing a complete solution to the general problem. In practice, this appeared to be a frustration for students as it made it hard to diagnose problems with their submissions as we report in Section 8. Limiting the number of times a student can submit solutions for a given task or assignment is intended to prevent students from gaming the system instead of carefully and thoughtfully updating their code.

3.3 Extracted dataset

We extracted the data from 1st to 4th year students stored by the PASTA system over the years of 2013, 2014 and 2015 in 13 computer science subjects, as depicted in Table 1. In particular, we consider five main metrics characterizing a student:

1. *performance* - the total mark (%) the student obtained for all assessments during the semester (*assessment performance*) and/or at the final exam (*exam performance*);

2. *readiness* - the time between the last student submission of the assignment and the assignment deadline, averaged over all assignments; a higher value means the final submission was made well before the deadline;

3. *earliness* - the time between the first submission of the assignment and the assignment deadline, averaged over all assignments; a higher value means that the student started working on the assignment earlier;

4. *programming language* - the programming language chosen for the submission of a particular assignment; note that Pasta only allows code submitted in C, C++, Java, MatLab and Python depending on the subject, and the same student could potentially choose a different language for each of their assignments in some subjects;

5. *effort* - the difference between the first and last assignment submission, in number of code lines that were changed or added, as measured by the `diff` comparison between the two submissions. The higher the difference, the higher the effort the student put in regardless of the average number of lines. Note that we did not consider the total number of lines of code produced before the use of the system.

For privacy reasons, we replaced the student identifiers in the obtained dataset with a MD5 hash. Note that student identifiers comprise letters taken from their firstname and lastname, which makes the student corresponding to a given entry easily identifiable when not anonymized in this way. We also excluded the information related to the submissions made by instructors, tutors and Pasta developers for testing and setup purposes. We then asked a person external to our project to replace the identifiable dataset by a new anonymized dataset on which we could run our analysis.

4. WHY INSTANT FEEDBACK?

In this section, we discuss potential benefits of instant feedback for students and instructors.

4.1 Formative instant feedback

The instant feedback delivered by our system is *formative* (as opposed to *summative*), that is, the information given to the learners is intended to modify their behaviour for the purpose of improving learning as defined by Shute [17]. Numerous studies have explored the benefits of formative feedback, and its positive influence on learning and motivation, with varying, sometimes contradicting findings. The excellent review on feedback by Shute analyses the underlying factors and proposes a set of guidelines to design feedback so that it is effective. In particular, instant feedback is advised for procedural or conceptual knowledge (e.g., instant feedback in programming tasks has been shown to produce greater learning gains [4], as well as for complex tasks, or when learners are struggling [17].

4.2 Feedback can impact student behaviour

To illustrate the effect of instant feedback, consider the two assignments of our "Distributed Systems and Network Principle" course, Comp2121. The first assignment is to write an SMTP server, due in week 5, whereas the second is to write a server for P2PTwitter, a dissemination protocol of a 140-character message to a peer-to-peer system, due in week 12. Instant feedback is provided for the first assignment but not for the second assignment. Despite the period during which students could submit their second assignment being longer than for the first assignment, we observe that students tend to submit during a larger period of time for the assignment with instant feedback as depicted in Figure 1. We can thus observe a different behaviour that may be attributed to the provision of instant feedback. It might be explained by the fact that (1) instant feedback helps students identifying their mistakes, (2) it motivates students to improve their work by providing them with an estimate of an expected mark. However, other factors could impact these results as well: for instance the fact that students behaviour can evolve from a first assignment to another or that the 2nd assignment was clarified to the students closer to the due date than the 1st assignment. In Section 8, we discuss how some of our students appreciated instant feedback.

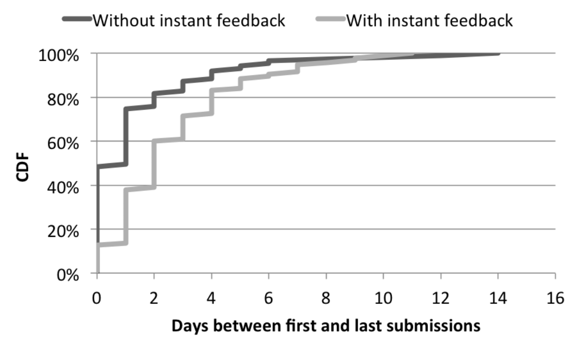


Figure 1: Cumulative Distribution Function (CDF) of the number of submissions per assignment in Comp2121

4.3 Feedback helps the instructor of non-programming courses

One advantage of providing instant feedback about compilation is that it greatly simplifies the task of the instructor during the marking process as students are motivated to correct and resubmit their programs, hence providing instructors with submissions that work and can be tested for their intended behaviour.

An example of a common error in Java is changing the organization of the source code folder upon submission of a Java program. Misplacing the classes within the packages will prevent the instructor from assessing easily the students even though the code is correct—this is typically a tolerable mistake in subjects that are not focused exclusively on assessing programming skills. As an example in Comp2121 (Semester 2, 2014), we observed 91 compilation errors out of 1123 submissions for the first assignment, accounting for 8%, and 41 compilation errors among 271 in the second assignment, accounting for 15% of the submissions. Note that the second value 15% is particularly insightful as there was no instant feedback (besides compilation errors) for motivating students to improve their submissions in the second assignment.

Another advantage resulting from saving human resources during the marking process is that instructors have more time to provide personalised and detailed feedback.

5. EFFORTS AND RESULTS

In this section, we analyse the students' effort on their

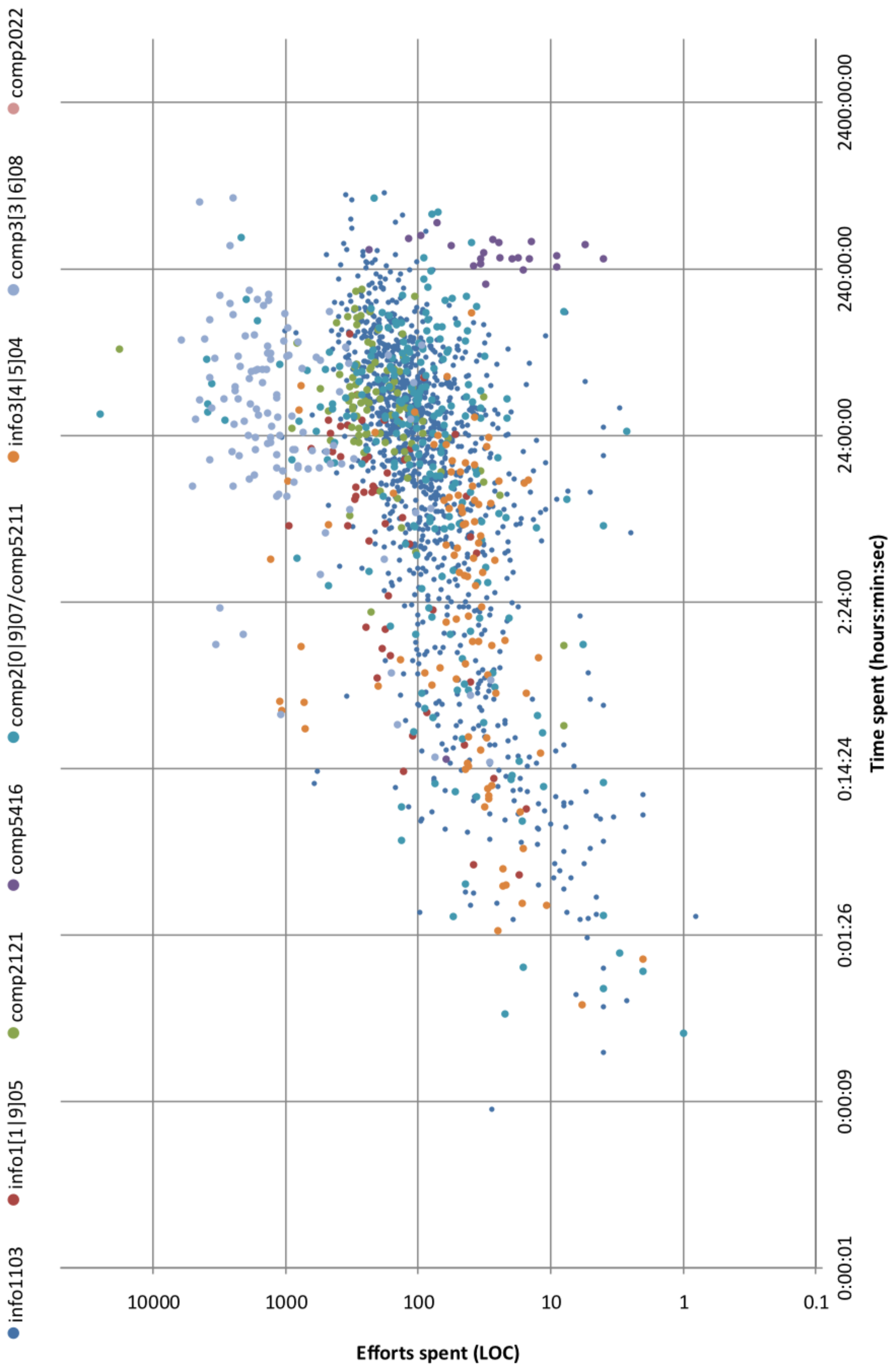


Figure 2: The efforts spent by lines of code (LOC) against the efforts spent in time between the first and the last submission times

assignments and the relation with the performance they obtained.

5.1 Student effort per assignment

Figure 2 depicts student effort on average per task. Tasks include practice sessions in labs as well as assignments. Note that we reduced the size of numerous INFO1103 points by half for the sake of visibility. Student effort is measured in terms of lines of code and time between first and last submissions. The LOC (lines of code) is the average over the number of lines of code appearing in the `diff` of the first submission and the last submission of all assignments. As expected, we can observe that the time between the first and last submissions is slightly correlated to the amount of code produced, indicating that student effort increases with the amount of time between first and last submissions.

5.2 Readiness and student performance

Figure 3 shows the relations between readiness, earliness and performance in COMP3308 and COMP3608. First, Figure 3(a) shows the relation between earliness and performance. The earlier students start submitting, the higher their assessment mark is. Figure 3(b) shows similarly that the earlier students stop submitting, the higher their assessment mark is. Finally, Figure 3(c) depicts the relation between earliness and readiness that seems to indicate that students starting early are also the one finishing early.

5.3 Analysis of variance

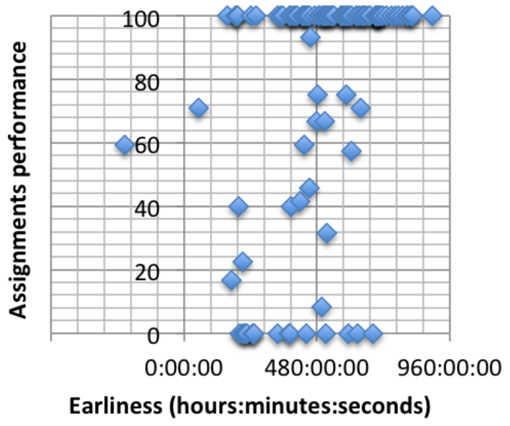
We performed a one-way analysis of variance (ANOVA) on the amount of time spent by highest performing students and lowest performing students, comparing the time spent by students that scored more than 50 against the students that scored less than or equal to 50 on the assignment. We could not observe a significant correlation. There might be several reasons for this, in particular, obtaining good marks at programming assignments does not necessarily result in also scoring good marks at the final exam, because final exams are typically broader and include theory questions. Since the final exam has a non-negligible weight in the overall performance, this may explain this lack of significance. In any case, more work is necessary to precisely assess the factors of student performance.

6. PROGRAMMING LANGUAGES

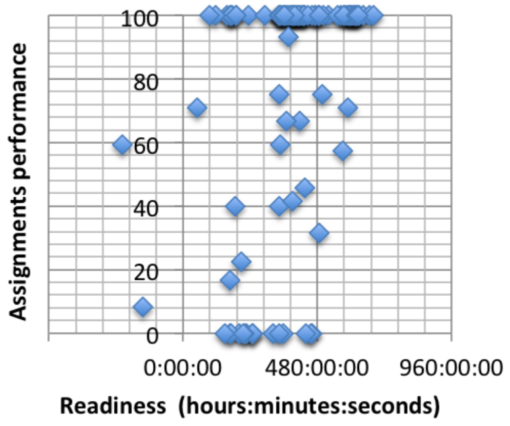
In courses that focus on assessing problem solving skills rather than programming language skills, it is common to offer students the choice of the programming language for their assignment. Therefore, we offered C, C++, Java, Python as supported programming languages in addition to Matlab in some 2nd, 3rd and 4th year courses.

6.1 Languages are not equally represented

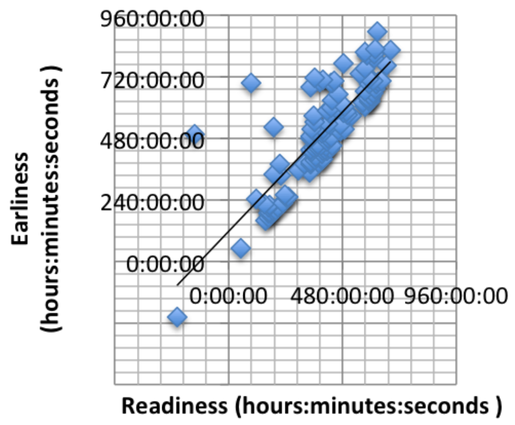
Figure 4 depicts the performance of students per programming language per subject. The languages are chosen for assignment submissions in courses COMP2007, COMP2907, COMP2022 and COMP5211 that relate to algorithms, logic and complexity, and are the only courses where students had the choice of the programming languages. We observed that students predominantly choose managed rather than unmanaged languages. In particular, they favour Java to code their assignment in algorithms and complexity courses. This may be explained by the fact that Java is taught in our



(a) Impact of earliness on performance



(b) Impact of readiness on performance



(c) Relation between readiness and earliness

Figure 3: Relation between efforts and performance in the Artificial Intelligence courses

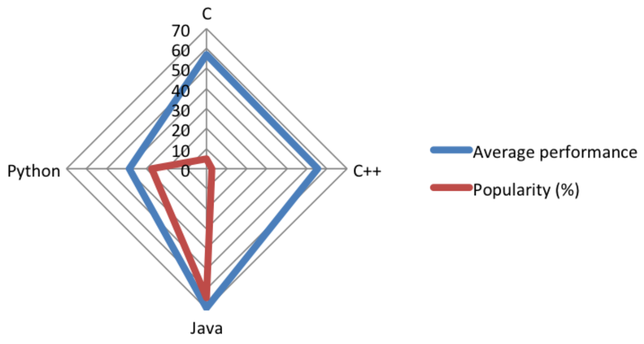


Figure 4: Popularity and performance per programming language

introduction to programming course.[1] An interesting observation is that some students vary their choice of languages. For example, in Comp2022 semester 1 of 2015, 12 students out of the cohort of 82 students used 2 different languages for submitting their assignments and we observed that all possible pairs of programming languages were selected by at least one student among this pool of students.

6.2 Languages are different across subjects

Figure 5(a) and 5(b) depict the performance of students at the final exam vs. the performance of students at the assignment. Each bubble represents one assignment per student. The different colours indicate the programming languages whereas the bubble size represents the number of submissions of a student for the corresponding assignment. First, we note that the choice of programming language does not impact the final exam mark. This is explained by the fact that this courses do not focus solely on assessing programming skills and that the final exam is also used to assess complementary skills, not assessed by the assignments.

Second, it is clear that C++ is rarely adopted by students and that Python and Java lead to very different performance. Students of Comp2007 who program in Python tend to have lower marks whereas those who program in Java tend to have higher marks. This confirms the comparison of average marks we observed on Figure 4. We observe the inverse phenomenon for students taking Comp2022 as they tend to have higher marks when they program in Python. One possible reason was that the performance of students in Comp2007 was assessed on the complexity of their algorithm in terms of running time. Despite assigning different weights to different languages, the complexity errors done by students in Python could have translated to a comparatively larger penalty than in Java. Finally, Java and C++ lead to higher average assignment marks than C and Python in algorithm courses, whereas Python leads to higher average assignment marks in the formal language and logic subjects.

7. DIFFERENT USES OF AUTOGRADING

We observed different usages across different courses. First the instructors of different courses are usually different, which may impact the way Pasta is setup. Second, the skills to be assessed are also different across subjects.

7.1 Test-driven development

For example, in the database courses (Info3404 and Info3504), students are exposed to several of the subsystems common to a relational database management system, and are introduced to algorithms (such as external merge sort and nested loops join) that seek to minimize the number of page accesses that typically dominate the latency of database operations. For the first half of the semester students are set weekly exercises to implement a particular algorithm, to be submitted via Pasta. This is intended to be training in a student's general programming skills, and in particular in test-driven development. Each week's task has several options of varying difficulty, with harder tasks worth more points. The exercises are all in Java, and tested with a set of JUnit tests. To avoid making the tasks too opaque, many of the unit tests are made available in source code.

7.2 Identification of specification violations

In distributed systems (Comp2121) or networks (Comp5416), students may have to implement a client and a server that communicate through a protocol, whether it is UDP, TCP, RTSP or a new one. For example, in the case the students submitting the server side, the test consists of running the student server code, and running clients against the server, sending requests to the server. This subjects often involved deployment and dedicated e-learning platforms have been proposed specifically for this purpose [10]. Testing protocols is typically very strict in that it requires the protocol specification to be respected. Pasta is useful to identify any sequence of commands within the protocol that is badly treated. Based on the feedback, the student can progressively nail down the command responsible for a bug and fix it. Sometimes, the test would require waiting for a server response. If the server does not respond before a predetermined period following a request, then the server would be considered violating the protocol. This delay may add up to the period the student has to wait before getting feedback.

7.3 Output and runtime analysis

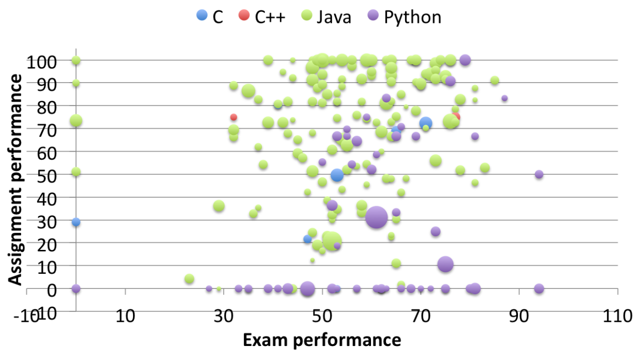
In some courses, the tests would consist of finding the difference between the correct expected output and the output of the student submission. In other cases, the runtime will be measured to deduce an estimate of the complexity of the algorithms. The timeout should be set depending on the programming languages used as they do not execute as fast as each other. Finally, some instructors often use Pasta during labs and not necessarily for student assessments.

8. STUDENT FEEDBACK

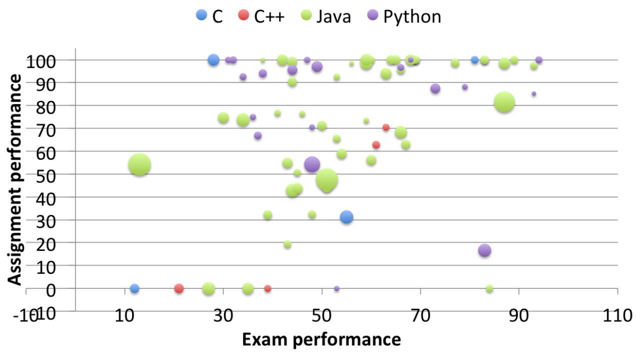
To better understand the perception of our students regarding the use of autograding and instant feedback in non-programming-focused subjects, we asked students taking the database course to answer a small questionnaire on a voluntary basis. 40 students out of 120 took the questionnaire, which had the following four questions:

- How have you used Pasta in the past?

[1]We considered files with extensions .cpp and .c as C++ and C files, respectively, without looking at the code itself.



(a) Comp2007: students tend to have lower marks when programming in Python for the Algorithms and Complexity course



(b) Comp2022: students tend to have higher marks when programming in Python for the Formal Language and Logic course

Figure 5: Student performance depending on language choices for programming assignments

- On balance, did you think Pasta helped you to learn?
- What was Pasta's worst features last time you used it?
- What was Pasta's best feature last time you used it?

8.1 General feedback

Out of the 40 students who answered, 9% of them used Pasta for the first time and 64% had found it helpful for learning. The predominant worst feature, identified by the 25% of the respondents, was the delayed response whereas the best feature, identified by the 39% of the respondents, was the feedback they received on the tests that were executed. The detailed results for the last two questions are depicted in Figure 6 as the percentage of responses that selected each feature. Note that the response percentages total to more than 100% as students could select multiple categories as worst features.

Pasta processes each student submission by adding it to a single processing queue and then waiting for the job to be run. This can lead to a delay in the student receiving the result for their latest submission, particularly during busy periods.

8.2 Hidden tests and limited submissions

Interestingly, students complained about the limited number of submissions and the hidden tests. These two parameters can make harder for students to improve their mark. As hidden tests are tests for which the system does not give any feedback: they typically force students to find the solution to their problem and test their code carefully on their own, without any support.

The limited number of submissions represents the maximal number of submissions a student was allowed to make for a given assignment. This number is a parameter set up by the instructor for each assignment and can be set to a predefined number or to $+\infty$. Unfortunately, we were unable to quantify these hidden tests and numbers of submissions as these were from the experience of students with Pasta in previous units.

8.3 Multiple submissions

While some students did not like having a cap on the number of submissions, students liked the fact that they had an opportunity to try again when a submission failed some tests ("Multiple Submissions" as best feature).

8.4 Instructor adjustments

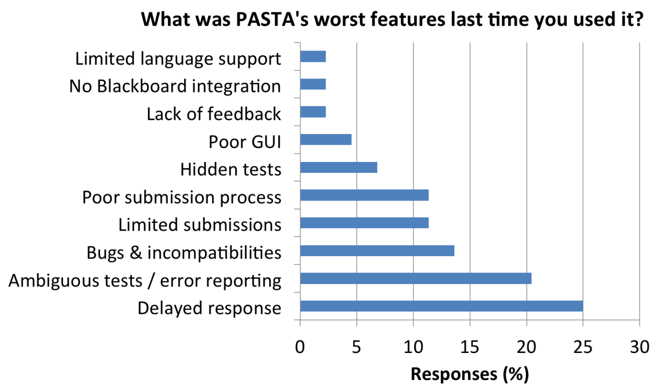
These responses helped some of the instructors to setup Pasta differently in the second semester of 2015. In particular, it was decided for Databases courses, Info3404 and Info3504, that the source code of the unit tests used by Pasta would be made available to students prior to submission. This allowed students to avoid the queueing waiting time of Pasta, to avoid reaching the cap of submissions allowed and to understand precisely ow the Pasta tests worked. Other instructors, however, proceeded differently. For example, in the Distributed Systems course, Comp2121, the instructor decided to provide the bytecode of the unit tests only, without the source code. While this also provided students with a solution to bypass the queueing waiting time of Pasta by testing locally, it prevented them from trivially implementing a solution over-optimized for the tests rather than for the more general problem.

9. DISCUSSION

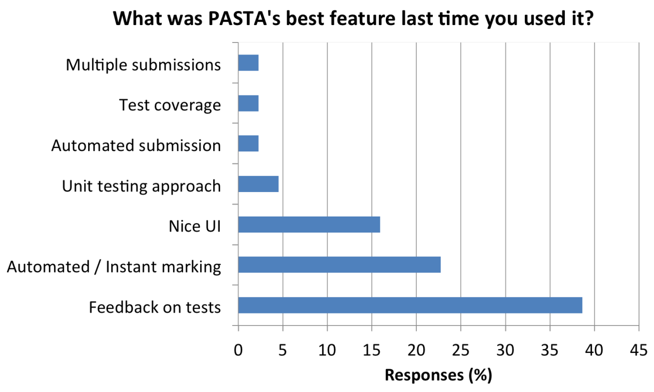
Our study consisted primarily of reporting the data collected during a period of three years while offering autograding and instant feedback to students in 13 diverse subjects. Our main conclusion is that such tools combine an instant feedback feature that is particularly useful for the students and an autograding feature that is useful for the instructors.

More specifically, we observed that this teaching approach is especially useful for instructors teaching subjects not focused on assessing programming skills. Although previous work already reported the benefit for students in programming subjects, our study outlines that this is beneficial to students studying subjects of computer science not necessarily focused on assessing programming skills.

Our study shows a strong relation between the time at which a student starts submitting programs to solve a task (earliness) and the time at which this student stops



(a) Worst features



(b) Best feature

Figure 6: Feedback of students regarding the Pasta autograding tool

(re)submitting programs for this task (readiness). We observed that students who start submitting early or stop (re)submitting early tend to get higher marks, however, this should be taken carefully as our results do not show a statistically relevant impact of earliness or readiness on the student performance. We do not consider that we have fully answered the question either: there are many factors that impact performance, and sometimes the performance at the final exam is not representative of the performance at programming assignments. The conclusions we drew from the extracted dataset are far from being complete, and we plan to continue our study with future collected data.

10. CONCLUSION

We presented an analysis of the impact of autograding and instant feedback in computer science education on students in 1st year to 4th year. The lack of grading model for assignments was identified as the drawback of existing tools [2] but the variety of subjects that will probably be semi-automatically assessed in the near future makes this task even more challenging.

We made the following observations. First, although instant feedback was shown to be useful for students in programming subjects, we showed that it also affects their behaviour in other subjects and we also observed that autograding is helpful for instructors not focusing on assessing programming skills. Second, student results varied significantly depending on the programming language they chose to code a programming task, and on the type of skills the programming task aimed at assessing. We observed that the same student may submit code written in different programming languages, even for different assignments of the same subject. Managed languages (e.g., Python, Java) are often preferred to unmanaged languages (e.g., C, C++). Finally, we observed a relation between the time at which a student starts submitting programs for a task and the time at which the student stops (re)submitting for this task.

We believe autograding and instant feedback could be useful to more advanced forms of programming courses as well. For example, the ACM/IEEE CS Curriculum 2013 contains a new knowledge area named Parallel and Distributed Computing [15] and multi-core programming has already been suggested as a topic in itself [13]. One could combine a framework like Pasta with Synchrobench [9] to assess students based on the performance of their concurrent data structures compared to the state-of-the-art.

Availability

The figures are available online in higher resolution at `https://sites.google.com/site/autogradinganalysis/`.

Acknowledgements

We are grateful to Greg Ryan for setting up the instant feedback tool we used in this study and to Josh Stretton for anonymizing our dataset. The research leading to this work was partially supported by the University of Sydney's Large-Scale Educational Innovation Grant 2015 entitled "Enhancing feedback in online assessment of computer programming". NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

11. REFERENCES

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

[2] Julio C. Caiza, Álamo Ramiro, and José María del. Programming assignments automatic grading: review of tools and implementations. In *Proceedings of the 7th International Technology, Education and Development Conference (INTED2013)*, pages 5691–5700, 2013.

[3] Brenda Cheang, Andy Kurnia, Andrew Lim, and Wee-Chong Oon. On automated grading of programming assignments in an academic institution. *Computers & Education*, 41(2):121 – 131, 2003.

[4] A. T. Corbett and J. R. Anderson. Locus of feedback control in computer-based tutoring: Impact on learning rate, achievement and attitudes. In *Proceedings of ACM CHI 2001 conference on human factors in computing systems*, pages 245–252, 2001.

[5] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. Codewrite: Supporting student-driven practice of java. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 471–476, New York, NY, USA, 2011. ACM.

[6] Stephen H. Edwards and Manuel A. Perez-Quinones. Web-cat: Automatically grading programming assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '08, pages 328–328, New York, NY, USA, 2008. ACM.

[7] E. Enstrom, G. Kreitz, F. Niemela, P. Soderman, and V. Kann. Five years with kattis—using an automated assessment system in teaching. In *Frontiers in Education Conference (FIE), 2011*, pages T3J–1–T3J–6, Oct 2011.

[8] Armando Fox. From MOOCs to SPOCs. *Commun. ACM*, 56(12):38–40, December 2013.

[9] Vincent Gramoli. More than you ever wanted to know about synchronization: Synchrobench, measuring the impact of the synchronization on concurrent algorithms. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP 2015, pages 1–10, 2015.

[10] Guillaume Jourjon, Salil Kanhere, and Jun Yao. Impact of an e-learning platform on cse lectures. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2011, pages 83–87, 2011.

[11] Irena Koprinska, Joshua Stretton, and Kalina Yacef. Predicting student performance from multiple data sources. In *Proceedings of the Artificial Intelligence in Education*, volume 9112 of *LNCS*, pages 678–681, 2015.

[12] Irena Koprinska, Joshua Stretton, and Kalina Yacef. Students at risk: Detection and remediation. In *The 8th International Conference on Educational Data Mining*, 2015.

[13] Timothy J. McGuire. Introducing multi-core programming into the lower-level curriculum: An incremental approach. *J. Comput. Sci. Coll.*, 25(3):118–119, January 2010.

[14] Dejan Milojicic. Autograding in the cloud: Interview with David O'Hallaron. *Internet Computing, IEEE*, 15(1):9–12, Jan 2011.

[15] The Joint Task Force on Computing Curricula. The ACM/IEEE computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science, Dec. 2013.

[16] Mark Sherman, Sarita Bassil, Derrell Lipman, Nat Tuck, and Fred Martin. Impact of auto-grading on an introductory computing course. *J. Comput. Sci. Coll.*, 28(6):69–75, June 2013.

[17] Valerie J. Shute. Focus on formative feedback. Technical Report RR-07-11, Educational Testing Service, Princeton, NJ, 2007.

[18] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 15–26, New York, NY, USA, 2013. ACM.

[19] Jaime Spacco, David Hovemeyer, William Pugh, Fawzi Emad, Jeffrey K. Hollingsworth, and Nelson Padua-Perez. Experiences with marmoset: Designing and using an advanced submission and testing system for programming courses. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITICSE '06, pages 13–17, 2006.

[20] Anne Venables and Liz Haywood. Programming students NEED instant feedback! In *Proceedings of the Fifth Australasian Conference on Computing Education (ACE'03)*, pages 267–272, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.