Logic, damned logic, and statistics

Andrew Simpson
Software Engineering Programme
Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD
United Kingdom
Andrew.Simpson@comlab.ox.ac.uk

Abstract

In this paper we present the results of a statistical analysis undertaken upon the performance of students within the Software Engineering Programme at Oxford. The Software Engineering Programme is aimed at part-time students, most of whom are professional software engineers. The Programme offers a total of 26 courses within the broad spectrum of software engineering, with six being in the formal methods area. The focus of the research undertaken for this paper was the Software Engineering Mathematics course, which provides an introduction to (Z-style) predicate logic, set theory, and proof. In particular, we concern ourselves with the issues of whether performance within the Software Engineering Mathematics course is a good predictor of performance overall.

Keywords: software engineering, education, formal methods

1. INTRODUCTION

The Software Engineering Programme at Oxford began in the early 1980s as a set of ‘industrial courses’: employees of organisations such as IBM would come to Oxford for an introduction to formal methods such as Z [23, 24] or CSP [13, 20]. An ‘integrated programme’ of six one-week courses was established in 1993; this has evolved into a comprehensive programme of education in software engineering, with 26 different courses (in subjects as diverse as Functional Programming, Web Services, and Managing Risk and Quality), and students from more than 100 different organisations.

In [6], the authors discussed how the Software Engineering Programme endeavours to teach formal methods ‘within context’—that is, by relating them directly to modern software engineering processes. In [21], the authors considered the relationship (previously discussed in [16]) between relational database design and the language of Z and explored how the relationship between the two paradigms is exploited within the teaching of the Programme. In [22], the authors discussed the supervision and assessment of projects within the context of the Software Engineering Programme. In this paper, we consider—from a statistical viewpoint—the relationship between the introductory formal methods course, Software Engineering Mathematics (which provides an introduction to the mathematical language of Z), and students’ performance in other courses as well as their performances overall.

Our reasons for undertaking this analysis are relatively straightforward. While the teaching of formal methods to undergraduate students needs clear motivation (see, for example, [18], [15], and [11]), a considered approach to links with other aspects of the curriculum (see, for example, [17] and [19]), thought with respect to assessment (see, for example, [14]), and, in some cases, approaches to compensating for limited mathematical backgrounds (see, for example, [3]) the challenges facing us in teaching formal methods to professional software engineers are slightly different. Our students are motivated—they wouldn’t be in the class otherwise; our students do attend lectures and classes—the mode of study means that they can’t avoid them; our students are (typically) mathematically able.

What we see as our main challenge is the demonstration that an appropriate application of formal methods is relevant to the professional software engineer in their every day activity—whether this involves reasoning about security protocols, implementing new modules, or writing SQL queries.

(Indeed, we view the demonstration that a ‘light touch’ application of formal methods is essential to demonstrating their industrial relevance.¹) It is to this end that we conducted our analysis.

The structure of the remainder of this paper is as follows. In Section 2 we provide an overview of the Software Engineering Programme at Oxford, and, in particular, discuss the provision for formal methods within the Programme. Then, in Sections 3 and 4 we list a number of questions that we set out to investigate within this research and discuss briefly the data that we have used. Next, in Section 5 we analyse our data with respect to the questions of Section 3. Finally, in Section 6 we summarise the contribution of this paper and indicate some potential next steps.

2. BACKGROUND

Both in the United Kingdom and in the United States there is, on the one hand, the need for a highly-skilled work force, and, on the other hand, a decline in public support for Higher Education [5]. In addition, as more young people enter Higher Education in the United Kingdom, the possession of a postgraduate degree is becoming a more significant factor in the job market—at the same time as the level of public funding for postgraduate taught courses is being reduced. This pattern is being replicated worldwide.

In [9], several themes that characterise higher education’s response to these problems are identified. Two of these themes—“a shift from faculty-centered to learner-centered institutions” and “lifelong learning”—are at the heart of the Software Engineering Programme at the University of Oxford.

There has also been a shift in Higher Education’s mission to include three additional goals: providing knowledge to the work-force, re-tooling people for new careers, and catering to the need for mental simulation [2]. Furthermore, universities are now being asked to consider what it means to be a graduate [12] and to debate the transferable skills that such graduates should possess [10].

The Software Engineering Programme at the University of Oxford is a joint venture between Oxford University Computing Laboratory and the University’s Department for Continuing Education, which has the aim of making the educational resources of the University of Oxford more accessible to a wider public. The Programme offers a number of short courses—which may be taken individually, combined to form short programmes of study, or used as credit towards a postgraduate qualification—that cover a wide range of software engineering topics.

At any one time, approximately 200 students are registered for a part-time qualification: either a Postgraduate Certificate (in Software Engineering, Object Technology, or Computer Security), a Postgraduate Diploma, or an MSc. These students (many of whom come from overseas) attend intensive, one-week courses at the Programme’s own teaching facility. The Programme currently has a core staff of sixteen: six university lecturers, four teaching assistants, one departmental lecturer, one career development fellow and one developer, all of whom are supported by an administrative staff of three.

The Programme allows students to register initially for one of three qualifications: a Postgraduate Certificate, a Postgraduate Diploma, or an MSc. As well as allowing students to move from registration for a lower qualification to a higher one, the Programme also allows students to move back down the ladder: many part-time students find themselves in the position of being unable to make sufficient time to complete their studies. It would be a pity for such students to leave the Programme without an award of any sort; hence, the possibility of transferring to a lower award.

To gain a Postgraduate Certificate, a student needs to attend and submit an assignment for four courses; to gain a Postgraduate Diploma, attendance and subsequent submission for eight courses is required; for an MSc, it is ten, together with the successful completion of a project and dissertation.

---

¹Note that by ‘light touch’ we do not mean the use of ‘lightweight’ methods as described in [4], but rather the *appropriate* use of techniques—possibly concentrating on a specific aspect of a system; possibly in conjunction with more informal techniques—within context.

Assignment submissions are awarded grades on a linear scale from $A+$ to $C-$. Grades in the range $A+$ to $A-$ are considered to be ‘distinction level’; grades in the range $B+$ to $B-$ are considered to be passing grades; grades in the range $C+$ to $C-$ are considered to be failing grades. It is worth noting that there is no notion of ‘passing’ an individual course: an average of grades is taken and it is this average that determines whether a student passes (if their average is $B-$ or above) or fails their award. It is also worth noting that some students will not submit an assignment: if this happens they may choose to take the assignment for the next instance of the course (students may also choose to take this course of action should they receive a failing grade, or, indeed, if they received a passing grade and wish to boost their average).

Formal methods are at the heart of the expanded programme. There are two courses that teach aspects of Z, two that teach CSP, one that teaches B [1], and another that is concerned with Functional Programming. A further course shows how Z and CSP can be used together (with UML, Java and JUnit) in the design, development, and testing of object-oriented software. Other courses adopt a similar, precise, and principled approach: rigorous modelling with UML; formal techniques for the analysis of security protocols; Z for database design; grammars for design patterns; formal descriptions of web services; precise process descriptions for service-based architectures; a formal approach to transformations in the context of XML; and notions of abstraction and refinement for model-based testing.

Our concern in this paper is the Software Engineering Mathematics course, which provides an introduction to the mathematical language of Z. Specifically, the course covers material from the first ten chapters of “Using Z” by Woodcock and Davies [24].

3. RESEARCH QUESTIONS

We considered the following questions in our analysis.

1. There is evidence (see, for example, [7] and [8]) that bimodal distributions are present in student populations in technical subjects such as programming and discrete mathematics. Is this the case here?
2. Do those students who take the opportunity to take assignments for subsequent courses improve their grades—or do some students simply never “get it”?
3. How does performance in Software Engineering Mathematics compare with students’ performances in other courses?
4. Is the Software Engineering Mathematics course a good predictor of overall performance?

We do not claim that the above collection of questions is in any exhaustive. Rather, these questions were chosen specifically because they help to explore commonly held viewpoints (or prejudices)—with a view to influencing future developments within the Programme.

4. DATA USED

We looked at grades for those candidates attending the last six deliveries of the Software Engineering Mathematics course (since June 2004). When those students taking the course on a ‘stand-alone’ basis (i.e., those students who attended the course (and possibly undertook and submitted the post-course assignment) but who weren’t registered for an award of the Programme) were removed from the population, we were left with a total of 53 students to work with. We also considered how those students performed on the other courses that they attended. The mathematical abilities of these students—as one would expect from students drawn from such a wide constituency—varies widely.

Note that for the purposes of all but the second question, we took the following viewpoint: if a student submitted assignments for two or more instances of the course, we took the highest of the grades. This was for the simple reason that it was exactly the view that would be taken by the examination committee.

There are, of course, some caveats to be inserted here. The first is the relatively small sample size; arguably, though, acquiring data from further back than June 2004 would mean that the relevance

of some that data might be brought into question (course provision changes; delivery style changes; etc.). Other fundamental caveats include: the nature of the Programme (being targeted at professional software engineers) is very different to an undergraduate programme in Computer Science; the nature of the teaching (in intensive one-week blocks) is different from the usual mode of delivery; and the nature of assessment (a take-home assignment, as opposed to an examination) differs from what most full-time students will be used to. Nevertheless, while some of the lessons learned will (somewhat inevitably) be primarily of interest to the Software Engineering Programme others may be symptomatic of experiences within more traditional environments, and, as a result, may be of interest to the wider community.

5. ANALYSIS

In this section we consider again the questions of Section 3.

5.1. There is evidence (see, for example, [7] and [8]) that bimodal distributions are present in student populations in technical subjects. Is this the case here?

Over the period, the following numbers of grades were awarded (recall that for candidates with two or more submissions, we count the best grade):

- $A+$: 8
- A : 12
- $A-$: 13
- $B+$: 7
- B : 4
- $B-$: 1
- $C+$: 1
- C : 2
- Did not submit: 5

There is certainly a normal(ish) hump at the top end. We can only speculate as to how the distribution at the bottom end would look if those candidates who failed to submit would have performed if they had submitted assignments. It may very well be that there is a normalish distribution of ability at the bottom end of the population, but the nature of our assessment means that we simply cannot tell. There is nothing here, though, to dispel the commonly held assumption that there are two distinct populations.

5.2. Do those students who take the opportunity to take assignments for subsequent courses improve their grades—or do some students simply never “get it”?

Surprisingly, we found that a difference emerges between those who did not submit and those who did submit but received a failing grade. Although the numbers are not significant, there was a definite trend: those that did not submit the first time also failed to submit the second time; those that submitted and received a failing grade the first time received a passing grade the second time. It is worth noting that detailed personalised feedback on submissions is received by students following all assignment submissions—this may well have a role to play in improving performance the second time around.

5.3. How does performance in Software Engineering Mathematics compare with other courses?

Software Engineering Mathematics is typically seen as a ‘hard’ course by students on the Programme—primarily because it represents a new way of thinking for many students. Yet the average grade for our student population in Software Engineering Mathematics is $A-$, whereas the average grade for the same population across their other courses is $B+$. To be precise, if we convert to a numeric scale, where $A+$ corresponds to 9, A corresponds to 8, etc., we find that the averages are 6.9 and 6.4 respectively. Also, 60% of students did better in Software Engineering Mathematics than they did in other subjects. Of course, the candidates taking the course are self-selecting; however, it is pleasing that this study has given rise to a means by which a commonly held perception amongst students can be countered.

5.4. Is the Software Engineering Mathematics course a good predictor of overall performance?

To tackle this question, we considered the subsequent performances of the last 15 students to either not submit an assignment or receive a failing grade for the course. This involved going back to data from the middle of 2003. The performances of the 15 students were as follows.

- 4 of the students are still registered with the Programme but have not submitted assignments for other courses since.
- 5 of the students are, to use a euphemism, 'struggling'.
- 2 of the students have since left the course.
- 2 of the students took the course at or near the end of the course, and, in both cases, the grades received made no material difference to their outcomes.
- 2 of the students took a subsequent assignment, improved the second time around, and are now progressing well.

This was perhaps the most surprising result of our study, but perhaps reflected something that we had subconsciously always suspected: that those students who do not have an aptitude for logic, set theory and proof tend to do poorly in the subject of Software Engineering as a whole.

6. CONCLUSIONS

The research being undertaken here is in no way exhaustive and is at a very early stage. However, our study has proved to be very useful: even if it is something that we might have anticipated, establishing that performances in Software Engineering Mathematics are a good predictor of future performance on the Software Engineering Programme overall helps to confirm some commonly held prejudices. Of course, not all students opt to take the Software Engineering Mathematics course—and some students who do well in the course might subsequently perform poorly overall—so it is an implication that has been established here, rather than an equivalence.

Our next step will be to tackle four further questions that are of specific interest to us. First, we intend to utilise the feedback forms from our formal methods courses to determine if those students who feel that the approaches and techniques are of relevance to their working life tend to perform better. Second, analysing performance in terms of students' background is also high on our agenda: does significant industrial experience have an impact on performance in the formal methods courses? Third, we'd like to be able to answer the question "does attendance on the Software Engineering Mathematics course result in improved performance in those courses (such as Database Design and Security Principles) where an understanding of discrete mathematics should, intuitively, help?" Fourth, we shall also be incorporating a simple 'before' and 'after' test into future iterations of the Software Engineering Mathematics course to determine the effectiveness of our teaching: will the students with an aptitude for the subject always do well? (And will those with little aptitude always do badly?) Finally—and more broadly—we intend to undertake a comparison with experiences within the full-time MSc in Computer Science at the University of Oxford to determine if there are any commonalities between the experiences of our students and those of 'regular' students.

REFERENCES

- [1] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.
- [2] H. Blustain, P. Goldstein, and G. Lozier. Assessing the new competitive landscape. In R. N. Katz and associates, editors, *Dancing With The Devil: Information Technology And The New Competition In Higher Education*, pages 51–72. Jossey-Bass inc., 1999.
- [3] E. Boerger. Teaching ASMs to practice-oriented students with limited mathematical backgrounds. In *Proceedings of Teaching Formal Methods 2003, Oxford Brookes University*, pages 5–12, 2003.
- [4] R. T. Boute. Can lightweight formal methods carry the weight? In *Proceedings of Teaching Formal Methods 2003, Oxford Brookes University*, pages 47–55, 2003.
- [5] D. W. Breneman, J. E. Finney, and B. M. Roherty. *Shaping The Future: Higher Education Finances In The 1990s*. San Jose: California Higher Education Policy Centre, 1997.

- [6] J. W. M. Davies, A. C. Simpson, and A. P. Martin. Teaching formal methods in context. In C. N. Dean and R. T. Boute, editors, *CoLogNET/FME Symposium, TFM 2004*, pages 185–202. Springer-Verlag Lecture Notes in Computer Science, volume 3294, 2004.
- [7] S. Dehnadi. Testing programming aptitude. PPIG Annual Conference, Brighton, 2006.
- [8] S. Dehnadi and R. Bornat. The camel has two humps. Little PPIG 2006, Coventry, 2006.
- [9] J. J. Duderstadt. Can colleges and universities survive in the information age? In R. N. Katz and associates, editors, *Dancing With The Devil: Information Technology And The New Competition In Higher Education*, pages 1–26. Jossey-Bass inc., 1999.
- [10] A. Griffin. Transferring learning in higher education: problems and possibilities. In R. Barnett, editor, *Academic Community: Discourse or Discord?* Jessica Kingsley, 1992.
- [11] H. Habrias and S. Faucou. Linking paradigms, semi-formal and formal notations. In C. N. Dean and R. T. Boute, editors, *CoLogNET/FME Symposium, TFM 2004*, pages 166–184. Springer-Verlag Lecture Notes in Computer Science, volume 3294, 2004.
- [12] HEQC. Assessment In Higher Education And The Role Of ‘Graduateness’. Higher Education Quality Council, London, 1997.
- [13] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.
- [14] S. King. The assessment of students on FM courses: a position paper. In *Proceedings of Teaching Formal Methods 2003, Oxford Brookes University*, pages 56–59, 2003.
- [15] M. Loomes, B. Christianson, and N. Davey. Formal systems, not methods. In C. N. Dean and R. T. Boute, editors, *CoLogNET/FME Symposium, TFM 2004*, pages 47–64. Springer-Verlag Lecture Notes in Computer Science, volume 3294, 2004.
- [16] A. P. Martin and A. C. Simpson. Generalizing the Z schema calculus: Database schemas and beyond. In *Proceedings of APSEC (Asia-Pacific Software Engineering Conference) 2003*, pages 28–37, 2003.
- [17] R. F. Paige and J. S. Ostroff. Specification-driven design with Eiffel and agents for teaching lightweight formal methods. In C. N. Dean and R. T. Boute, editors, *CoLogNET/FME Symposium, TFM 2004*, pages 166–184. Springer-Verlag Lecture Notes in Computer Science, volume 3294, 2004.
- [18] J. N. Reed and J. E. Sinclair. Motivating study of formal methods in the classroom. In C. N. Dean and R. T. Boute, editors, *CoLogNET/FME Symposium, TFM 2004*, pages 32–46. Springer-Verlag Lecture Notes in Computer Science, volume 3294, 2004.
- [19] K. Robinson. Embedding formal development in software engineering. In C. N. Dean and R. T. Boute, editors, *CoLogNET/FME Symposium, TFM 2004*, pages 203–213. Springer-Verlag Lecture Notes in Computer Science, volume 3294, 2004.
- [20] A. W. Roscoe. *The theory and practice of concurrency*. Prentice-Hall, 1998.
- [21] A. C. Simpson and A. P. Martin. Supplementing the understanding of Z: a formal approach to database design. In *Proceedins of Teaching Formal Methods 2003, Oxford Brookes University*, pages 65–70, 2003.
- [22] A. C. Simpson, A. P. Martin, J. Gibbons, J. W. M. Davies, and S. W. McKeever. On the supervision and assessment of part-time postgraduate software engineering projects. In *Proceedings of the 25th International Conference on Software Engineering (ICSE)*, pages 628–633. IEEE Computer Society Press, 2003.
- [23] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall International, second edition, 1992.
- [24] J. C. P. Woodcock and J. W. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall International, 1996.