Canonical Explorations of 'Tel' Environments for Computer Programming

Richard Mather, Buckinghamshire New University, United Kingdom
Nicholas Day, Buckinghamshire New University, United Kingdom
Richard Jones, Buckinghamshire New University, United Kingdom
Carlo Lusuardi, Buckinghamshire New University, United Kingdom
Kevin Maher, Buckinghamshire New University, United Kingdom
Barbara Dexter, Buckinghamshire New University, United Kingdom

The European Conference on Technology in the Classroom
Official Conference Proceedings 2015

Abstract
This paper applies a novel technique of canonical gradient analysis, pioneered in ecological sciences, with the aim of exploring student performance and behaviours (such as communication and collaboration) while undertaking formative and summative tasks in technology enhanced learning (TEL) environments for computer programming. The research emphasis is, therefore, on revealing complex patterns, trends, tacit communications and technology interactions associated with a particular type of learning environment, rather than the testing of discrete hypotheses. The study is based on observations of first year programming modules in BSc Computing and closely related joint-honours with software engineering, web and game development courses. This research extends earlier work, and evaluates the suitability of canonical approaches for exploring complex dimensional gradients represented by multivariate and technology-enhanced learning environments. The advancements represented here are: (1) an extended context, beyond the use of the 'Ceebot' learning platform, to include learning-achievement following advanced instruction using an industry-standard integrated development environment, or IDE, for engineering software; and (2) longitudinal comparison of consistency of findings across cohort years. Direct findings (from analyses based on code tests, module assessment and questionnaire surveys) reveal overall engagement with and high acceptance of collaborative working and of the TEL environments used, but an inconsistent relationship between deeply learned programming skills and module performance. The paper also discusses research findings in the contexts of established and emerging teaching practices for computer programming, as well as government policies and commercial requirements for improved capacity in computer-science related industries.

Keywords**:** Technology-Enhanced Learning; Computer Programming; Collaborative Learning.

Introduction

Ever since the introduction of FORTRAN to Higher Education in the 1960s (Davey & Parker, 2010), computer programming has been considered a challenging subject to learn. Although much progress has been made with teaching techniques, 50 years on, students still struggle to learn the underlying concepts and to apply principles to solve programming problems.

Programming modules are central to the four computing courses taught here at Buckinghamshire New University (hereafter referred to as 'Bucks'). Due to the considerable revenues that digital and technological industries attract to the UK economy (Sparrow, 2006) the newly elected government has reaffirmed a commitment to policy to promote "high levels of skills in science, technology, engineering and maths (STEM), and citizens that value them" (UK Government Department for Business Innovation & Skills, 2015). The number of programming and software engineering related jobs are growing at a rate that is faster than most other sectors. In the USA, some forecasts estimate growth between 20-30% by 2020 (Bureau of Labor Statistics, 2014; Zhao, 2013), thus indicating a continued global demand for skilled programmers. Livingstone and Hope (2011), co-author of "The Next Gen" report, however found that there had been a reduction in the numbers of qualified graduates entering the video games industry. The authors attributed this to university curricula that "neglected" the computer science and programming skills needed by this sector (Gove, 2012; Livingstone & Hope, 2011).

In Higher Education, although the dropout rate amongst other subjects has decreased recently (HEFCE, 2013), this is not the case with computing and computer science courses. Programming modules are often cited as an obstacle to progression. It is estimated that between "30% (in the 1960s) and 50% (some institutions in 2010s)" of students taking programming will fail or dropout (Bornat, 2011). This is consistent with the experiences at Bucks where many students are required to retake introductory programming modules in order that they may progress to second year studies.

This research continues the work of Mather (2015), presented at the IAFOR ECTC conference in 2014. Research reported here endeavours to determine whether the patterns of engagement and learning progress reported this year are consistent with those reported for 2014, and thereby indicate whether earlier observations are likely to be representative of future cohorts. Since Mather's earlier study, assessment criteria of the modules involved were altered in an attempt to encourage the development of a deeper understanding of programming skills required by industry. In the 2013-2014 academic year students were required to complete and document all tasks (numbering 100 or so problems) issued in 'study packs'. These comprise some 5-7 in-class practical exercises and 1-3 independent studies each week. In a further week exercises are replaced with a single in-depth and extended project. This academic year (2014-2015) students were requested to only submit independent studies and projects, significantly reducing the volume of work required for assessment. It was hoped that this would allow greater time to concentrate on

completing tasks to a higher standard. The course team also introduced voluntary formative testing measures to help students consolidate subjects covered in theory and practical sessions.

Aim and Objectives

This paper applies a novel technique of canonical gradient analysis, pioneered in ecological sciences, with the aim of exploring student performance and behaviours (such as communication and collaboration) while undertaking formative and summative tasks in technology-enhanced learning (TEL) environments for computer programming.

Research emphasis is, therefore, on revealing complex patterns, trends, tacit communications and technology interactions associated with a particular type of learning environment, rather than the testing of discrete hypotheses. The study is based on observations of first year programming modules in BSc Computing and closely related joint-honours computing with software engineering, web and game development courses. This research extends earlier work, and evaluates the suitability of canonical approaches for exploring complex dimensional gradients represented by multivariate and technology-enhanced learning environments. The advancements represented here are: (1) an expanded context, beyond the use of the 'Ceebot' learning platform, to include learning-achievement following advanced instruction using an industry standard integrated development environment, or IDE, for engineering software; and (2) longitudinal comparison of consistency of findings across cohort years.

The overall aim was to continue applying the technique of 'canonical' analysis to explore student progress and patterns of engagement behaviours (including communication and collaboration) while using in-house TEL environments for learning programming.

The supporting objectives were as follow:

- To suggest behaviours associated with learning success;
- By validating studies reported in 2014, to indicate whether behaviours may reliably predict learning success (thereby, have potential value as early signals for remedial intervention);
- To determine whether the modified assessment regime has improved the alignment of assignments to workplace skills;
- To compare the findings of 2015 with those reported in 2014 with respect to the persistence (thereby predictability) of engagement patterns (regardless of the use of an additional learning platform and modified regime);
- To comment on the usefulness of canonical approaches in the light of extended findings.

Literature Review

Many educational researchers have attempted to discover predictors for success in learning programming, with varying results. The following review suggests that learning success cannot be attributed to simple educational factors or solutions. This perhaps reflects the multifaceted nature of learning and its dependence on complex interactions between diverse influences such as teaching methods, learning styles, supporting materials, modes of assessment and the learning environment. The review, however, attempts to identify prominent student behaviours and/or attributes that may either contribute to success or obstruct progress when learning programming. These factors may influence the delivery of programming courses and guide teacher intervention.

Dehnadi and Bornat (2006) claimed to have designed an aptitude test able to predict "with high accuracy" the students most likely to succeed when learning programming at an introductory level. Their aptitude test contained 20 questions on the assignment of variables (entities used to store information/values in programs), and was administered to 61 students in both the first and third weeks of module delivery. According to the answers submitted, students were placed in one of three 'mental model' categories that represented the conceptual approaches that non-programming students used to solve problems. The categories were 'consistent' (those using the same conceptual model), 'inconsistent' (those drawing from a variety approaches) and 'blank' (those not answering most or all questions).

In their analyses of two sets of exam results, Dehnadi and Bornat (*ibid.*) demonstrated that the 'consistent' group significantly outperformed their 'inconsistent' peers. Although the test was replicated by other researchers (Caspersen *et al.,* 2007; Ford & Venema, 2010), these did not demonstrate similar associations between performance and 'mental model', thus placing the prediction value of Dehnadi's and Bornat's (*op cit.*) in some doubt. Richard Bornat (2014) has since modified claims concerning the reliability of their test as a result of findings from subsequent administrations of the test. Nevertheless, he continues to advocate the need for students to develop consistent mental models in order to succeed as programmers. Other authors have identified specific topics that cause confusion when learning programming. These include problems relating to determining the values assigned to variables (e.g. du Boulay, 1986; Bayman & Mayer, 1983; Perkins & Simmons, 1988), and related problems working with iteration constructs that programmers use to repeat code (Bornat, 2011; Kessler & Anderson, 1986).

The late Steve Jobs (1995) remarked on the importance of programming as a 'mirror for thought processes' explaining that "everyone should learn how to program a computer, because it teaches you how to think". This is consistent with Ben-Ari (1998) who associates the educational philosophy of constructivism with learning how to program. Students create their own mental models of a concept, based on their previous knowledge and experiences, rather than 'copying' someone else's idea of what the concept is, from a book or from the lecturer. Biggs and Tang (2011) stress the importance of constructively aligning the course materials, teaching, and

assessment so that students have opportunities to coherently relate ideas and concepts from these individual elements of the course. Hagan and Markham (2000) found that students who had prior experience in a programming language performed "significantly better" in assessment. They surveyed students at four stages during an introductory programming module at Monash University in Australia. Data collection was by questionnaire survey and included biographical information, educational expectations and programming experience. The authors were therefore also able to observe that greater grade performance was associated with prior programming experience and to exposure to more than one programming language. It is of further interest that other researchers have found that prior experience can be a hindrance if this based on misconceptions or incorrect models of programming concepts (Bonar & Soloway, 1985; Taylor & Du Boulay, 1987; Lui *et al.*, 2004). Conversely, Longo (2010) regards that the competitive advantage of prior experience is unaffected if students already possess some 'pre-existing matrix', or schema, or correct mental model of programming principles.

Following a study that questioned students on their understanding of passages of prose, Marton and Saljo (1976) famously reported that individuals either adopted "surface-level or deep-level processing" strategies. Surface learning behaviour is generally associated with a focus on knowing what needs to be known in order to complete a task. In contrast, deep learners consider the underlying reasons behind concepts and explore these in greater detail. Bornat (2011) suggests that the decision to learn lies with the student; in other words teaching does not necessarily result in learning.

Simon and co-authors (2006), reporting on a "multi-national and multi-institutional" study in 2004, found that grade results were positively correlated with a deep approach to learning and negatively correlated with adopting a surface approach. The study compiled the data of 177 participants from 11 institutions based in Australia, New Zealand and Scotland. The study comprised four tests: a spatial visualisation task (a standard paper folding test); two behavioural tasks - drawing a simple map, and articulating a search strategy; and an attitudinal task based on a questionnaire that requested students to provide details of their approaches to learning and studying.

It is also commonly suggested that there is an association between mathematical and programming ability. Wilson and Shrock (2001) conducted a study that recorded twelve 'predictors' for 105 students. They found that a background in mathematics was the second strongest predictor of success in a programming module. Van der Veer and various colleagues (1983; 1986) also report that students who are strong in maths do not require as much teaching as other students. Simon and co-workers *(op. cit.*), similarly observe a correlation between maths ability and programming ability, and suggest that the two disciplines require similar logic to step through problems. Here at Bucks, there is anecdotal evidence that students who are strong in maths can solve problems faster than those who are not. Mather (*op. cit.*) also found that students were able to solve problems within the environment that they were being taught in, but many struggled to answer questions outside that environment:

suggesting the underlying concepts were not learnt to a level whereby they could be confidently transferred to other environments.

Programming modules are typically structured so that students build on knowledge each week, whereby concepts introduced later in the module often require that foundation principles are properly understood. This is sometimes referred to as 'scaffolded' learning (Sawyer, 2006) and is also consistent with Meyer & Land's (2013) notion that many subject disciplines have 'threshold concepts'. Such thresholds are critical transformations in understanding that are necessary for a learner to progress to more advanced topics.

Robins (2010) noticed that students built momentum as they succeeded in learning the first topic, and then found it easier to learn the second, and then the third and so on. However, students who failed to learn foundation topics often found it difficult to progress further. The loss of momentum at early stages may also signal reduced motivation and a weakening in engagement and attendance.

Given the difficulties that students experience when learning programming, it seems probable that such reduced levels of motivation may, in some part, be attributable to low 'self-efficacy'. This is described as "the personal perception of one's ability to successfully execute an activity" (Ponton & Rhea, 2006 after Bandura, 1997). Ponton and Rhea (2006), who contextualise autonomous learning in the Social Cognitive Theory of Bandura (1986), consider how three key agents (environment, the learner and behaviour) reciprocally influence the development of self-efficacy through 'mastery experiences'. These are experiences where the learner genuinely attributes 'successes' to personal ability. Conversely, successes borne of 'environmental factors', such as assistance from others or "information supplied by the environment", do not enhance self-efficacy (Ponton & Rhea, *op. cit*. after Bandura, 1997).

Whilst academic skills are usually the subject of research, academic ability is also significantly affected by wellbeing and personal circumstances. Wilson & Shrock (*op. cit.*), observed that 'comfort level' was the most significant predictor of student success in a programming course. Outside of programming, many authors have commented on the negative impact that anxiety has on students (Medlicott, 2009, Richardson *et al.*, 2012). The pressure to do well has always existed in academic contexts, but in many countries this anxiety is now exacerbated by increased tuition fees against a background of global recession and reduced opportunities for employment. De Raadt and colleagues (2005) note a change in research focus from 'presage factors' (IQ, previous programming experience) to more holistic views surrounding issues of welfare, experience and emotional circumstances that may influence capacity for learning.

Students respond differently when faced with programming problems. Perkins and co-workers (1989) noticed that some students were easily frustrated with problems and quickly abandoned attempts at solutions. These students were categorised by the authors as 'stoppers'. Conversely, other students (categorised as 'movers')

recognised when they were unable to solve problems, and would progress by working on other tasks to make more effective use of time. As well as being effective in time management, 'movers' were less prone to lose motivation than 'stoppers'.

Motivation is frequently cited as a key factor in determining student success across many disciplines. Geoff Petty (2009) recognises this to be one of the most significant challenges faced by teachers and students alike, and of particular consequence for the "digital native" generation (Prensky, 2001; Ritchel, 2010). This generation is often characterised, even stereotyped, by behavioural attributes that include: low attention span; prone to distractions and continually switching between tasks; having insufficient 'downtime' to connect ideas; lack of sleep due to time spent with social media etc.

Many authors have found that the best predictor of success revolves around student expectations, especially of grades (Richardson *et al*., 2012; Rountree *et al*., 2002), but that attitude, keenness and general academic motivation are also important (Simon *et al*., 2006).

Methods

The approach adopted was similar to that described by Mather (*op. cit.*), thereby ensuring that findings could form part of a longitudinal study. This adopted an ecological perspective that, similar to the positioning of organisms along natural environmental gradients, the position that students may occupy along achievement gradients is partly determined by their learning behaviours and their engagement with learning environments.

Mather's 2014 questionnaire was slightly amended. The overall structure remained unchanged as follows:

- an initial question asking students to evaluate the 'perceived difficulty' of modules;
- five test coding questions, four requiring coding concept definitions and one to identify output from a 'broken' code;
- 22 Likert scale questions (20 in 2014) concerning learning behaviours and module acceptance attitudes.

Likert scale questions were modified in 2015. This was because in the earlier study (Mather, *ibid*.) the combined test and questionnaire was issued after students had completed only one of two programming modules, this based on the 'Ceebot' learning environment. In 2015, the test and survey was conducted at the end of the academic year when students had almost completed both programming modules. Questions were therefore amended to capture impressions of both Ceebot (Semester 1 module) and the C# language using Microsoft's Visual Studio™ platform (Semester 2 module). A further question was included to determine student preferences for learning programming with either Ceebot or the Visual Studio™ integrated development environment (IDE).

This research forms part of the teaching team's ongoing evaluation of module delivery. Full student consent was obtained and findings have been anonymised.

Results and Discussion

5.1 The application and interpretation of Redundancy Analysis

Questionnaire data was transferred to a spreadsheet. Programming test data was scored for correctness and 21 opinion questions captured using five Likert categories. For the purposes of canonical analysis (see Mather 2015 for details of this procedure), independent variables (corresponding to 'environmental gradients'), were represented by programming skill and knowledge and module grade. The dependent variables were represented by the Likert opinion responses and a percentage record of student attendance. The analysis was conducted using the canonical analysis software Canoco 5™. This is widely used by ecologists to conduct multivariate analyses and reveal patterns and trends in data. Here redundancy analysis (RDA) is applied to the data set. Among key advantages of RDA over more commonly used ordination techniques (such as Principal Components Analysis) are:

- that ordination axes are constrained in iterative steps to describe variation in the explanatory variables of interest;
- the technique does not require assumptions of unimodality (ter Braak, 1987);
- the resulting correlation biplots provide an easily interpretable and graphical summary of the most important relationships in the ordination model.

The reader is again referred to Mather (*op. cit.*) and the original and updated works of Canoco specialists (e.g. Corsten & Gabriel, 1976; ter Braak, *op. cit.* and 1992; Šmilauer & Lepš, 2014) for more detailed description of the mathematical interpretation of vectors represented by RDA biplots.

For the purposes of visual interpretation of the biplot figures below it is sufficient to understand that vectors or 'arrows' point in the direction of maximum variation. Variables with longer arrows have greater effect on the overall model and are generally most closely correlated with the independent variables of interest (ter Braak, 1987; ter Braak &Prentice, 1988). Variables and axes pointing in the same direction are positively correlated, perpendicular vectors are uncorrelated and opposing ones are negatively correlated.

With respect to negative correlations, these may only be artefacts caused by 'negative' assertions in expressing questionnaire items. In the biplots below independent variables are indicated by red arrows and dependent ones by blue arrows.

The summary statistics for analyses of both 2014 and 2015 data are presented in Table 1. Eigenvalues and other expressions for the proportion of variation explained by the first two (most important) axes, as well as for overall models, are slightly greater in 2014 than in 2015. This is perhaps due to the inclusion of further variables that were later shown to be rather weakly related to explanatory variation (notably two questions to capture preferences for the two environments used for teaching). Monte Carlo permutation tests indicate that model axes significantly describe variation in dependent variables.

Table 1 Summary statistics for RDAs presented in Figures 1 and 2.

Statistic	*Axis 1*	*Axis 2*
2015 Analysis		
Eigenvalues	0.0770	0.0258
Explained all variation (cumulative %)	7.70	10.27
Pseudo-canonical correlation	0.5968	0.6222
Explained fitted variation (cumulative %)	66.98	89.40
Permutation Test Results (on all axes)	pseudo-*F*=1.5; *p*=0.05	
2014 Analysis		
Eigenvalues	0.0915	0.0516
Explained all variation (cumulative %)	9.15	14.31
Pseudo-canonical correlation	0.8523	0.6975
Explained fitted variation (cumulative %)	54.04	84.52
Permutation Test Results (on all axes)	pseudo-*F*=1.4; *p*=0.04	

5.2 Results for 2015 and their comparison with 2014

Despite changes made to delivery, as well as differences in student cohorts and assessment regime in 2015, the overall patterns of biplot ordination for 2015 (Figure 1) were similar to those in 2014 (Figure 2).

The most notable feature is that in 2015 (similar to 2014) the alignment between module assessment and 'real' programming skills ("Module %" and "Test …" vectors in Figures 1 and 2), is not as strong as the programing team would wish. In fact, the increased angle (towards the perpendicular) between module assessment and test variables suggests that the modified assessment may not have had the desired effect of cultivating a deeper understanding of principles.

With regard to other relationships, all assessment indices (red) appear to remain (as in 2014) relatively strongly correlated with the 'commitment' indicators of independent study and homework (16, 19), but less so (in 2015) with the inclination to maintain a logbook of practical work (as indicated by the shorter vector for item 12 in 2015 compared to 2014). This latter observation is of some pedagogic significance because the modified assessment regime for 2015 also omitted the need to submit in-class solutions, and required students to present a smaller portfolio of independent-study tasks.

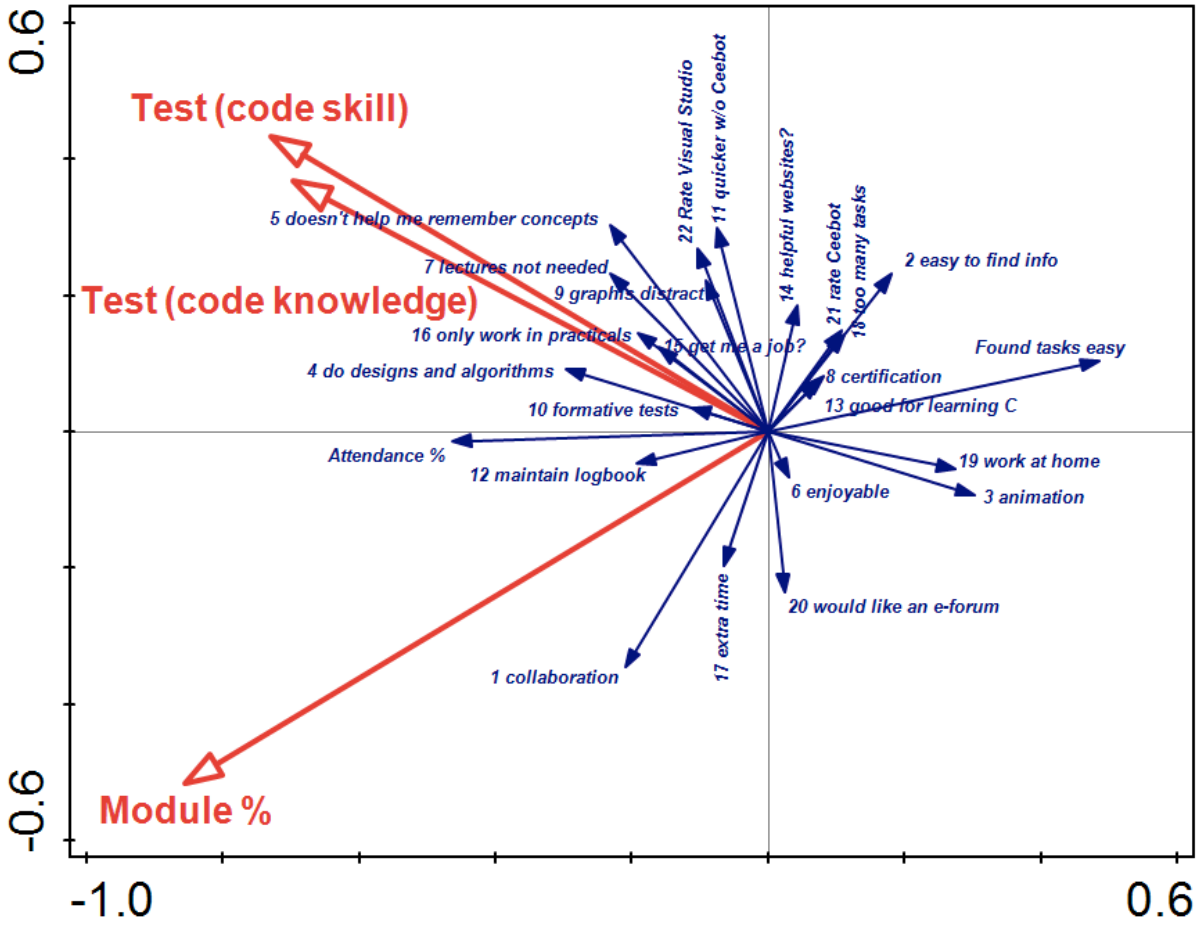

Figure 1: Biplot for RDA of 2015 data. Note: response variables (student behaviour and acceptance items captured by questionnaire and attendance logs) are represented by blue arrows and with three key 'explanatory' learning performance gradients (red arrows) for Module % and tested programming knowledge and coding skill.

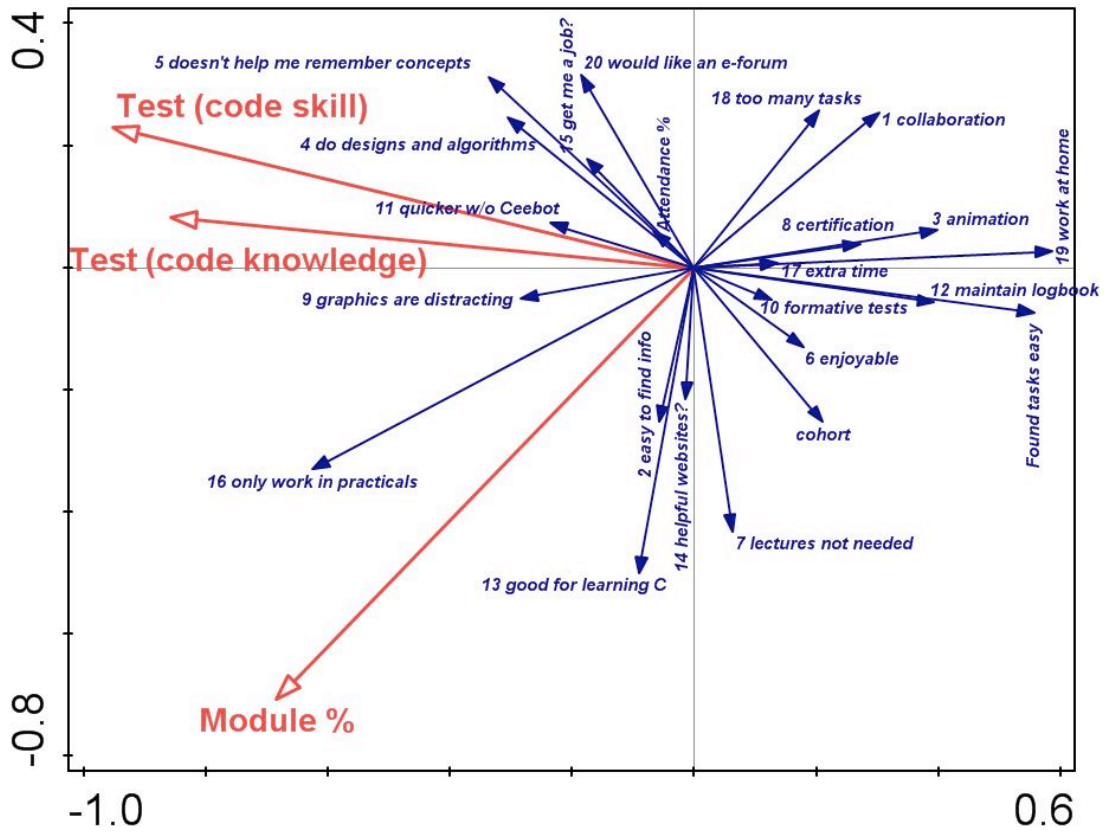

Figure 2. Ordination biplot from RDA of 2014 for comparison with 2015 (reproduced from Mather, 2015). Note: see caption for Figure 1 for interpretation.

Other noticeable between-year pattern similarities included both the orientation and strength (length) of key variables. These included expressions: “found tasks easy” (easy=scored 1 and very difficult scored 4); item 3, Ceebot animation was helpful (1=strongly agree to 5 strongly disagree); and item 19 preference to “work at home” (1=strongly agree to 5 strongly disagree). In both years these three variables are relatively strong and closely aligned with the first axis (indicating their overall importance in the model). Their opposing orientation to assessment and test variables is only an artefact of the direction of Likert category. Thus those who found tasks easier (probably through greater practice), appreciated the animation used in the Ceebot learning environment and were inclined to work outside lessons, predictably, achieving greater module and test scores. It was also unsurprising to discover that in both years, successful students had a tendency to “do designs and algorithms” (4) for their solutions; a behaviour which also directly contributes to module grades.

Of remaining similar and influential variables, the relationships between test score vectors and the item 5 assertion that Ceebot “doesn’t help me remember concepts” (and to lesser extents disagreements that it is “quicker to learn without Ceebot” and “only work in practicals”, items 11 and 16), are reassuringly consistent across both years.

Over interpretation of weaker relationships is perhaps best avoided on the grounds that any model influences are unlikely to be significant. These include expressions for enjoyableness (item 6), the desirability of certification (8), a need to have extra time (17), the desirability of including formative tests (10), relevance for employment (15), and ease in discovering help (items 2 and 14). Similarly, analysis of questionnaire items introduced in 2015 suggests a slight preference for using Microsoft Visual Studio™ over Ceebot. It may also be inferred from the direction of relationships, that Visual Studio™ is associated with higher code skill scores, and preference for Ceebot is more strongly associated with assessment success. However, although the patterns may be consistent with such views, neither relationship is confidently demonstrated by the analysis.

Regarding differences between years, in 2015 'attendance' is clearly more important than 2014 and is also more closely aligned with the three learning indicators. This may be attributed to a small but significant cohort of unusually capable game development students who, in 2014, were able to complete programming work on a self-directed basis while only attending classes infrequently.

5.3 A changed 'polarity' for student collaboration – is this an unintended consequence of the reduced assessment of in-class work?

Although clearly visible in both models, the changed polarity for collaboration (item 1) is less easy to explain. In both 2014 and 2015 collaboration (the full questionnaire expression for this item was in fact "While working on exercises it is very helpful to discuss problems with friends") is closely aligned with overall assessment success (Module %). In 2014 the opposition of collaboration and assessment vectors are consistent with a rationale that a high desire to discuss problems with colleagues (agreement with item 1 therefore low score) is associated with module success. In 2015, this relationship is such that low desire to collaborate is associated with success. It may be suggested that the 2015 group was less 'cohesive' than in 2014 (attendance and overall module success indicators are in fact consistent with this view), but such a complete reversal in direction of relationship indicates a need to more fully investigate the role of collaboration in learning programming.

One interesting (and pedagogically important) scenario is that by completely removing classwork from assessed elements in 2015, rather than having a desired impact of encouraging students to more deeply explore a smaller number of solutions, this has substantially reduced the need for collaborative and discursive interaction towards discovering programming solutions.

6. Conclusion and recommendations

This extension to the work of Mather (2015) demonstrates the overall usefulness of canonical RDA as means for exploring student progress in the educational contexts addressed by this study. Findings also were found to be useful for revealing patterns of engagement with learning materials and the class environment, as well as associating these with measures of learning achievement.

In 2015, as was the case in 2014, certain 'commitment' behaviours (such as the willingness to undertake homework and other independent study) are consistently associated with 'success'. The consistent relationship of these and other 'behaviours' across both years suggests their reliability as 'indicators' of success as well as of acceptance and engagement with learning environments.

The continuing orthogonality of the relationship between module grade and skill test data code (despite changes made to assessments) is a matter of ongoing concern to the teaching team. Additionally, the teaching team will wish to further investigate the possibility that the reduced in-class component of assessment (intended to allow greater time to develop deeper subject understanding) may have inadvertently encouraged 'surface' strategies to complete independent studies at the expense of equally important (but now unassessed) collaborative classwork. It may also be concluded that analyses did not demonstrate any improvement in the alignment of assignments to workplace skills as a result of modifications to the assessment regime.

Many aspects of the observations reported here are consistent with findings reported in literature by the wider research community. Although no causality is demonstrated by analyses here, variation in module achievement, nevertheless reflects the widely ranging circumstances of the student body (such as prior knowledge, 'comfort' in the academic environment, expectations and motivation) that are known to influence success (e.g. Richardson *et al*., 2012; Rountree *et al*., 2002; Simon *et al*., 2006; Wilson & Shrock, 2001).

Overall many of the indicators used in our findings signal the importance of fully engaging with studies at all levels (attendance, collaboration, self-directed study, participation with formative class work as well as summative independent study). This is consistent with wide acceptance that deep-level processes are required to embed learning (Marton & Saljo, 1976) and that computer programming is no exception to this principle (Simon *et al*., 2006),

Acknowledgements

The authors acknowledge the contribution made by first year students attending BSc Computing, Computing and the Web, Games Development and Software Engineering courses, who consented and participated in this research. We also acknowledge Bucks New University's support in funding research.

References

Bandura, A. (1986). *Social foundations of thought and action: A social cognitive theory.*
Englewood Cliffs, New Jersey: Prentice Hall.

Bandura, A. (1997). *Self-efficacy: The exercise of control.* New York: W. H. Freeman.

Bayman, P. & Mayer, R. E. (1983). A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements. *Communications of the ACM*, **26**(9), 677-679.

Ben-Ari, M. (1998) Constructivism in computer science education, *Journal of Computers in Mathematics and Science Teaching*. In ACM SIGCSE, **30**(1), 257-261.

Biggs, J. & Tang, C. (2011) *Teaching for Quality Learning at University*. OUP: McGraw Hill.

Bonar, J. & Soloway, E. (1985). Preprogramming knowledge: a major source of misconceptions in novice programmers*. Human-Computer Interaction* **1**, 133-161.

Bornat, R. (2011) *Some problems of teaching (learning) first-year programming (plus a glimmer of hope)*. Available at: http://www.researchgate.net/profile/Richard_Bornat/publication/266342475_Some_problems_of_teaching_%28learning%29_first-year_programming_%28plus_a_glimmer_of_hope%29/links/54b79bd20cf24eb34f6ebfbf.pdf (accessed July 2015).

Bornat, R. (2014) *Camels and humps: a retraction.* Available at: http://www.eis.mdx.ac.uk/staffpages/r_bornat/papers/camel_hump_retraction.pdf (accessed July 2015).

Bureau of Labor Statistics (2014) *Occupational Outlook Handbook: Software Engineers.* Available at: http://www.bls.gov/ooh/Computer-and-Information-Technology/Software-developers.htm (accessed May 2015)

Caspersen, M. E., Larsen, K. D. & Bennedsen, J. (2007). Mental models and programming aptitude. *ACM SIGCSE Bulletin*, **39**(3), 206-210.

Corsten, L.C.A. & Gabriel, K.R. (1976). Graphical exploration in comparing variance matrices. Biometrics 39, 159-168.

Davey, W. & Parker K.R. (2010). Turning Points in Computer Education. In A. Tatnall (Ed), *History of Computing. Learning from the Past*. Springer, pp.159-168.

de Raadt, M., Hamilton, M., Lister, R., Tutty, J., Baker, B., Box, I., Cutts, Q., Fincher, S., Hamer, J., Haden, P., Petre, M., Robins, A., Simon, Sutton, K. & Tolhurst, D. (2005).

Approaches to learning in computer programming students and their effect on success. In Brew, A., & Asmar, C. (2005). Higher Education in a changing world: *Research and Development in Higher Education,* **28**, 407-414.

Dehnadi, S. & Bornat, R. (2006) *The camel has two humps*. Available from: http://www.eis.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf (accessed July 2015).

Du Boulay, B. (1986). Some Difficulties of Learning to Program. *Journal of Educational Computing Research*, **2**(1), 57-73.

Ford, M. & Venema, S. (2010) Assessing the Success of an Introductory Programming Course. *Journal of Information Technology Education*, **9**, 133-145.

Gove, M. (2012) *Michael Gove at BETT 2012.* Available at: https://www.youtube.com/watch?v=Lwyawf91ITo (accessed June 2015).

Hagan, D. & Markham, S. (2000). Does it help to have some programming experience before beginning a computing degree program? *ACM SIGCSE Bulletin* **32**(3), 25-28.

Higher Education Funding Council for England (HEFCE), (2013), Non-continuation rates at English HEIs: Trends for entrants 2005-06 to 2010-11. Available at: http://www.hefce.ac.uk/pubs/year/2013/201307/#d.en.81697 (accessed July 2015).

Jobs, S. (1995) *Steve Jobs: the lost interview*. Available at: https://www.youtube.com/watch?v=TRZAJY23xio (accessed May 2015).

Kessler, C. & Anderson, J. R. (1986). Learning flow of control: Recursive and iterative procedures. *Human Computer Interaction*, **2**, 135-166.

Livingstone, I. & Hope, A (2011) *Next Gen. Transforming the UK into the world's leading talent hub for the video games and visual effects industries* [online]. Available from: http://www.education.gov.uk/ta-assets/~/media/get_into_teaching/resources/subjects_age_groups/cs_next_generation.pdf (accessed July 2015).

Longo, F. (2010) *Learning and memory: how it works and where it fails* [online]. Available from: https://www.youtube.com/watch?v=a_HfSnQqeyY (accessed July 2015).

Lui A. K., Khan R., Poon M. & Cheung Y. H.Y (2004) Saving weak programming students: applying constructivism in a first programming course*.* *inroads – The SIGCSE Bulletin* **36**(2) pp 72-76.

Marton, F. & Saljo, R (1976) On qualitative differences in learning — 2: outcome as a function of the learner's conception of the task. *Brit. J. Educ. Psych.* **46**, pp 115-27.

Mather R.A. (2015) Multivariate Gradient Analysis for Evaluating and Visualizing a Learning Platform for Computer Programming. *IAFOR Journal of Education,* **3**(1), pp 17-30.

Medlicott, D. (2009) How to design and deliver enhanced modules: a case study approach. Open University Press. pp 33.

Meyer, J., & Land, R. (2013). *Overcoming barriers to student understanding: Threshold concepts and troublesome knowledge*. London, Routledge.

Perkins, D. N., & Simmons, R. (1988). Patterns for Misunderstanding: An Integrative Model for Science, Math, and Programming. *Review of Educational Research*, **58**(3), 303-326.

Perkins, D.N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1989). *Conditions of learning in novice programmers*. In E. Soloway & J.C. Spohrer (Eds.) Studying the novice programmer (pp. 261-279). Hillsdale, NJ: Lawrence Erlbaum.

Petty, G. (2009). *Teaching today. A practical guide.* Nelson Thornes Limited.

Ponton, M.K., & Rhea, N.E. (2006) *New Horizons in Adult Education and Human Resource Development*, **20**(2), 38-49.

Prensky, M. (2001). Digital natives, digital immigrants. In *On the Horizon* (MCB University Press, **9**(5), October 2001).

Richardson, M., Bond, R., & Abraham, C. (2012) Psychological correlates of university students' academic performance: a review and meta-analysis. *Psychological bulletin 2012*, **138**(2), pp 353-382. Available from: http://emilkirkegaard.dk/en/wp-content/uploads/Psychological-correlates-of-university-students-academic-performance.pdf (accessed July 2015).

Ritchel, M. (2010). Growing up wired for distraction. Available at: http://www.nytimes.com/2010/11/21/technology/21brain.html?_r=4&sq=growing%20up%20digital&st=cse&adxnnl=1&scp=1&pagewanted=1&adxnnlx=1431430787-VNnzd76N+PtbUZ4KAwDfag (accessed July 2015).

Robins, A. (2010) Learning edge moment: a new account of outcomes in CS1, *Computer Science Education* **20**(1), 37-71.

Rountree N, Rountree J, & Robins A (2002). Predictors of success and failure in a CS1 course*. inroads SIGCSE Bulletin* **34**(2), 121- 124.

Sawyer, R. Keith. (2006). *The Cambridge Handbook of the Learning Sciences.* New York: Cambridge University Press.

Simon, Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., de Raadt, M., Haden, P., Hamer, J., Hamilton, M., Lister, R., Petre, M., Sutton, K., Tolhurst, D., and Tutty, J. (2006) Predictors of success in a first programming course. In Proc. Eighth Australasian Computing Education Conference (*ACE2006*), pp 189–196.

Šmilauer, P. & Lepš, J. (2014) *Multivariate Analysis of Ecological Data using CANOCO 5*. Cambridge University Press.

Sparrow, E. (2006) *Developing the Future. A Report on the Challenges and Opportunities Facing the UK Software Development Industry,* Microsoft Corporation.

Taylor, J. & du Boulay, B. (1987). *Learning and using Prolog: an empirical investigation.* Cognitive Science Research Reports No. 90. Brighton: University of Sussex.

ter Braak, C.J.F. (1986) Canonical correspondence analysis: a new eigenvector technique for multivariate direct gradient analysis. *Ecology* **67**(5), 1167-1179.

ter Braak, C.J.F. (1987) Ordination. In Jongman, R.H., C.J.F. ter Braak and O.F.R. van Tongeren, editors. *Data Analysis in Community Ecology*. Pudoc, Wageningen, The Netherlands.

ter Braak, C.J.F. (1992) *CANOCO - a FORTRAN program for Canonical Community Ordination*. Microcomputer Power. lthaca. New York.

ter Braak, C.J.F. & Prentice, I.C. (1988) A theory of gradient analysis. *Advances in Ecological Research* **18**, 271-317.

UK Government Department for Business Innovation & Skills (2015) *Policy paper: 2010 to 2015 government policy: public understanding of science and engineering.* Available at: https://www.gov.uk/government/publications/2010-to-2015-government-policy-public-understanding-of-science-and-engineering/2010-to-2015-government-policy-public-understanding-of-science-and-engineering (accessed July 2015).

van der Veer, G. & van der Wolde, J. (1983). Individual differences and aspects of control flow notations. In T.R.G. Green, S.J Payne & D. van der Veer (Eds), *The Psychology of Computer Use*. Academic Press (London).

van der Veer, G., van Beek, J. & Gruts, G.A.N. (1986). Learning structured diagrams. Effects of mathematical background, instruction and problem semantics. *Visual Aids in Programming*. Passau.

Wilson, B.C. & Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. *ACM SIGCSE Bulletin* **33**(1):184-188.

Zhao, G. (2013), *On effective pedagogical practice in teaching computer programming to CS/CIS majors*. Available at: https://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCcQFjAAahUKEwjv947NoO7GAhWDbRQKHfRABjk&url=http%3A%2F%2Ffiles.figshare.com%2F1027557%2FGrace_Zhao_NDSI_2013.pdf&ei=fUuvVa_FHoPbUfSBmcgD&usg=AFQjCNGeqUDKpMqrIsihovjxkmQXWE3DLA&sig2=hXzrr1tp98Gkxtj2TGo_Nw (accessed July 2015).