A Flexible Approach to Introductory Programming

Engaging and motivating students.

Neil Gordon[†]
Computer Science
University of Hull
Hull, UK
n.a.gordon@hull.ac.uk

Mike Brayshaw
Computer Science
University of Hull
Hull, UK
m.brayshaw@hull.ac.uk

Simon Grey
Computer Science
University of Hull
Hull, UK
s.grey@@hull.ac.uk

ABSTRACT

In this paper, we consider an approach to supporting students of Computer Science as they embark upon their university studies. The transition to Computer Science can be challenging for students, and equally challenging for those teaching them. Issues that are unusual – if not unique – to teaching computing at this level include

- the wide variety in students background, varying from no prior experience to extensive development practice;
- the positives and negatives of dealing with self-taught hobbyists who may developed buggy mental models of the task in hand and are not aware of the problem;
- the challenge of getting students to engage with material that includes extensive practical element;
- the atypical profile of a computing cohort, with typically 80%+ male students.

The variation in background includes the style of prior academic experience, with some students coming from traditional level 3 (i.e. A-levels), some through more vocational routes (e.g. B-Tech, though these have changed in recent years), through to those from experiential (work based) learning. Technical background varies from science, mathematical and computing experience, to no direct advanced technical or scientific experience.

A further issue is students' attainment and progression within higher education, where the success and outcomes in computer science has been identified as particularly problematic. Computer Science has one the worst records for retention (i.e. students leaving with no award, or a lower award than that originally applied for), and the second worst for attainment (i.e. achieving a good degree, that being defined as a first or a 2:1).

One way to attempt to improve these outcomes is by identifying effective ways to improve student engagement. This can be through appropriate motivators – though then the balance of extrinsic versus intrinsic motivation becomes critical. In this paper, we consider how to utilize assessment – combining the formative and summative aspects - as a substitute for coarser approaches based on attendance monitoring.

CCS CONCEPTS

• Computing Education • Computer-managed instruction • Computer Science education • Human Computer Interaction

KEYWORDS

Flexible Pedagogy; Introductory Programming; Gamification; Student Engagement.

CEP '19, January 9, 2019, Durham, United Kingdom
© 2019 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6631-1/19/01…$15.00
https://doi.org/10.1145/3294016.3294025

1 Introduction

A key component of computing courses is that of developing the ability to program [1], [20]. However, achieving this is a challenge [22]: Woodfield [25] identified computer science as one of the worst performing UK disciplines, in terms of both attainment and retention. As a discipline that includes both theory and application, the ability to carry out practical application through programming means that students must be able to translate the theory and skills they learn into actual practice as we aim to develop computational thinking [24]. Where practice can include motivational activities, it has the potential to improve student engagement in their studies, and thereby to potentially improve their attainment.

Computer Science is atypical to most subjects, in that the experience prior to university can vary from a decade of programming – with algorithmic thinking and coding now a part of primary education [16] – to those with no experience of computing including programming itself. As a follow up to Woodfield's report, Gordon [14] identified a range of approaches that are specific to computing and that can support students in their studies; of particular relevance to programming are those of using pedagogic styles that encourage engagement (gamification techniques, developing student communities and peer support, and using assessment to direct student focus).

2 Engagement indicators and measures

Student engagement in higher education is a commonly used term, though can be more difficult to define and measure [1]. For Computer Science education, where classes can be large (300+) and resources limited, encouraging student engagement and identifying those who are not engaging is critical. Attendance monitoring is occasionally used as a proxy for engagement [18], though our investigation of outcomes shows that this is not particular effective. Moreover, in terms of motivating the desired activities in our students, the use of extrinsic motivation – such as attendance monitoring – may have other unwanted effects, whilst failing to engender the desired impact on learning [17].

3 Flexible Learning and Technology

Technology can enable a more flexible approach to learning and assessment, providing mechanisms to allow students to gain more control over their pace, place and mode of learning. These 3-dimensions of flexibility [13] are particularly relevant in the context of teaching computing, where large, disparate cohorts with differing academic backgrounds and levels of engagement can benefit from tailored learning and support. As noted earlier, the challenge of large classes means that individual teaching can be limited. Equally, the varied background means that assuming the same level of progress is inappropriate.

Flexible pedagogy [23] is concerned with enabling (some level of) student choice. Technologies such as virtual learning environments, interactive learning tools and suitable subject specific tools (programming tutors) can enable students to have (some level of) choice [13].

4 Programming tutor systems

Developing the skill of programming raises a number of challenges. One is what high-level language, and correspondingly what programming environment, to use. At school and university level, early approaches to programming may be based on visual (block based) programming [21], such as Scratch, Blockly or the BBC Microbit language. Another choice is whether to look to interpreted languages (Basic, Python) or compiled ones (which may be include intermediate languages, as with Java and C#). A key issue here is the initial size of the story – in terms of the narrative of the virtual machine - that you are trying to tell the neophyte programmer [3], [8]. The bigger the story the more they have to learn and therefore the bigger is the conceptual task and the more chances there are of developing misconceptions, leading to loss of confidence and motivation. In choosing a language, many have a strong paradigm that can create barriers to initial learning (for example, with OOP languages, the size of the story of the virtual machine and the OO syntax and requirements can be barriers to students understanding the more fundamental programming concepts). The Development Environment themselves can also provide a barrier to students: with the complexity of many modern IDEs creating a steep learning curve.

One approach is to let pedagogy be used to drive the flexibility. Discovery learning allows the learners to find their explanations and solutions [5]. They find it in their own language and as an extension to their then mental model of the problem [18]. However just letting this happen as a natural process is slow, haphazard, and prone to meandering and dead ends and thus leading to a demotivating experience. Elsom-Cook [9] [10] [11] proposed to make this more efficient via the notion of Guided Discovery Learning. This aims to keep all the openness and flexibility of discovery learning but with the subtle guiding of the user through the process, keeping close to the point and avoiding unwanted diversions. Butterworth and Brayshaw [4] developed a programming tutor, building on earlier work that identified the benefits of visualizing the functionality of a program and emphasizing the transparency and story behind the virtual machine [9]. Butterworth and Brayshaw provided a scaffolded approach to programming, with skeletal code fragments and directed activities to limit the scope of students. They aimed to meet the meet the flexibility of discovery learning with suitable guiding on the journey. By enabling students to progress through activities in a structured way, they could identify their own individual progress, and be offered appropriate challenging levels (akin to moving through the levels in a game) as they completed activities.

5 A framework for teaching programming

We now explain this flexible approach to programming, with attendance monitoring replaced by students engaging with the teaching resources. The framework proposed in Figure 1 shows the scope of the range of tools intended to support the teaching of programming. These can be categorized into four main areas as:

1. Student Interface: a web browser interface for the student to interact with the code, as well as the working copy of the code being worked upon;
2. A source control host for the student code (in the example we use subversion, though other systems such as GIT would be equally viable);
3. Tutor services: control over the problems and access to code examples
4. Tutor Interface: edit the activities, link activities to learning outcomes, configure gamified aspects and review progress.

At this point, various aspects have been implemented and utilized. These include the use of source control (SVN) to identify student behaviours and track specific difficulties, as well as an approach to scaffolding the learning so that students can focus on particular aspects of programming that the teacher identifies as critical. The primary aim is to improve student engagement, with the intended consequence of improving their development of computational thinking.

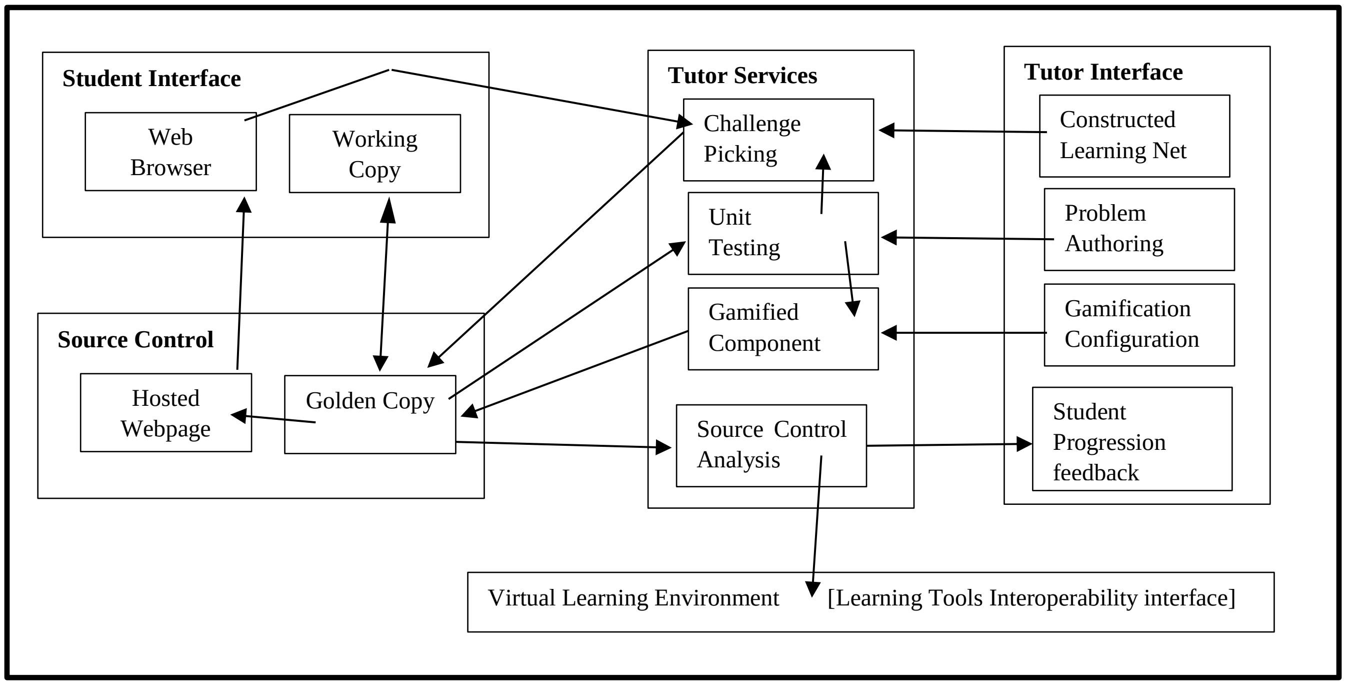

Figure 1: Framework for teaching programming

5.1 What is it?

The overall framework identifies a collection of tools and systems that provide students with programming tasks and activities. Code management tools (such as subversion) are used to track student interaction with the work, and to gauge their level of engagement. The programming tutor tool provides an interface and scaffolding for the specific programming work that the students need to do. The intention is to remove the barriers that a full IDE and the full syntax of an OOP language such as C# can create, thus allowing students on the introductory programming course to focus on the key programming concepts, such as conditional statements or iteration. The tool provides a mechanism for students to learn to program, without the distractions of artefacts of the language syntax and the IDE.

A key feature of this approach is the use of interaction with the staged activities as a better catalyst for engagement, than extrinsic motivators such as attendance monitoring. The programming tutor thus encourages students to focus on the relevant programming concept and to do so on a regular basis.

5.2 Why are we doing it?

The earlier discussion identified some of the rationale for this approach: as we seek to improve student engagement, to remove the barriers of unnecessary (at that point) language syntax (e.g. class constructs in OOP), and the complexity of a fully functioning IDE.

Furthermore, traditional approaches to teaching programming were based on the lecture, workshop, lab model. In one sense, this model of teaching reflects a science based “flipped learning” model, where the lecture is used to provide the material, ready for the workshop discussion and then practical (lab) application. Some of the problems that the traditional model faces are that

a) student attention and active engagement in a lecture can be limited;
b) workshops and labs are both relatively labour intensive, whilst the students who need the most help may not come forward to ask for it, or be recognized as needing that help by the staff

5.2 Where does it fit?

This framework and case study is based on the first year of computer science at our institution, based on our experience of a cohort of anywhere from 160 to 300 students. Given the challenges of the mixed background, the cohort is streamed into those with evidence of a fair level of programming knowledge, and those with a lower degree of programming, i.e. effectively novice programmers. The framework and tool are primarily targeted at these novice programmers.

5.3 Does it work?

Early evaluation of the use of source control shows this can be a better gauge of student engagement than attendance, since the activity closely aligns with the desired behaviours of a software developer. The programming tutor tool is still under evaluation as we are currently using it, but early appraisal shows that students are effectively engaging with the materials as they attempt to complete the programming activities presented within the tutor. The student interface connects to the source control system to provide the initial golden copy of code, and then enables students to work on their own copy.

6 Conclusions and next steps

This paper has considered some of the challenges in teaching introductory programming. We have provided a framework for supporting students in this transition, along with some proposals for providing a flexible structure to enable students to learn at their own pace and in a style, which suits them. This approach utilises both a pedagogic and a technological framework, with a flexible and gamified pedagogic methodology, scaffolded by the interconnecting technologies of source control and engaging interactive interfaces. Next steps for this work are a formal assessment of the impact on student engagement and learning, to assess the effectiveness of this in terms of

- motivation and engagement: potentially improving student interaction with learning materials, their active participation in workshops;
- learning outcomes: enabling students to become more proficient programmers;
- discipline skills: improving the development of computational thinking..

REFERENCES

[1] Association of Computing Machinery. 2005. Computing Curricula 2005.
[2] Axelson, R.D. and Flick, A., 2010. Defining student engagement. Change: The magazine of higher learning, 43(1), pp.38-43.
[3] Borning A. and O'Shea T. 1987 Deltatalk: An Empirically and Aesthetically Motivated Simplification of the Smalltalk-80 Language. In: Bézivin J., Hullot JM., Cointe P., Lieberman H. (eds) ECOOP' 87 European Conference on Object-Oriented Programming. ECOOP 1987. Lecture Notes in Computer Science, vol 276. Springer, Berlin, Heidelberg
[4] Brayshaw, M., Gordon, N., Nganji, J., Wen, L. and Butterfield, A., 2014, June. Investigating heuristic evaluation as a methodology for evaluating pedagogical software: an analysis employing three case studies. In International Conference on Learning and Collaboration Technologies (pp. 25-35). Springer, Cham.
[5] Bruner, J.S., 1961, The Act of Discovery, Harvard Educational Review, **31**(1), 21-3.
[6] Butterfield, A.M., and Brayshaw, M., A Pedagogically Motivated Guided Discovery Tutoring System for C#, Proceedings of the HEA STEM (Computing) Learning Technologies 2014 Workshop, University of Hull; 01/2014. https:www.researchgate.net/publication/263213711_Proceedings_of_the_HEA_STEM_%Computing%29_Learning_Technologies_2014_Workshop.
[7] du Boulay, B., O'Shea, T, Monk, J. The black box inside the glass box, International Journal of Human-Computer Studies – Special issue: 1969-1999, the 30th Anniversary, **51**(2), Aug 1999, pp. 265-277, Academic Press, Duluth, MN, USA.
[8] Eisenstadt, M., 1983, A user-friendly software environment for the novice programmer, Communications of the ACM, **12**(12).
[9] Eisenstadt, M., and Brayshaw,M., 1988, The Transparent Prolog Machine (TPM): An execution Model and graphical debugger for logic programming, **5**(4), pp. 277-342.
[10] Elsom-Cook, M., 1984, Design Considerations for an intelligent tutoring system for LISP, PhD Thesis (Unpublished), Department of Psychology, University of Warwick, UK.
[11] Elsom-Cook, M., 1990a, Guided Discovery Tutoring, in M Elsom-Cook, *Guided Discovery Tutoring: A Framework for ICAI Research,* London: Paul Chapman.
[12] Elsom-Cook, M., 1990b., Extended Computer-Aided Learning Minimalism in Guided Discovery Learning, in M Elsom-Cook, *Guided Discovery Tutoring: A Framework for ICAI Research,* London: Paul Chapman.
[13] Gordon, N. (2014). Flexible pedagogies: Technology-enhanced learning. From the report series Flexible Pedagogies: Preparing for the Future. The Higher Education Academy, January. Online at: http://www. heacademy.ac.uk/resources/detail/flexiblelearning/flexiblepedagogies/tech_enhanced_learning/main_report (accessed 20 June 2014).
[14] Gordon, N. "Issues in retention and attainment in Computer Science." York: Higher Education Academy (2016).
[15] Gordon, N., Brayshaw, M. and Grey, S., 2013. Maximising gain for minimal pain: Utilising natural game mechanics. Innovation in Teaching and Learning in Information and Computer Sciences, 12(1), pp.27-38.
[16] The Guardian, 2014. Coding at school: a parent's guide to England's new computing curriculum. https://www.theguardian.com/technology/2014/sep/04/coding-school-computing-children-programming
[17] The Guardian, 2018. Why are students faking attendance? They feel cheated by the system. Online: https://www.theguardian.com/higher-education-network/2018/apr/13/why-are-students-faking-attendance-they-feel-cheated-by-the-system.
[18] Nyamapfene, A., 2010. Does class attendance still matter?. Engineering Education, 5(1), pp.64-74.
[19] Papert, S., 1980, *Mindstorms: Children, Computers and Powerful Ideas*, Brighton:Harverster Press.
[20] Quality Assurance Agency. 2016. Subject Benchmark for Computing.
[21] Ray, P.P., 2017. A survey on visual programming languages in internet of things. Scientific Programming, 2017.
[22] Rogerson, C. and Scott, E., 2010. The fear factor: How it affects students learning to program in a tertiary environment. Journal of Information Technology Education: Research, 9, pp.147-171.
[23] Ryan, A. and Tilbury, D., 2013. Flexible Pedagogies: new pedagogical ideas. Higher Education Academy, London.
[24] Wing, J.M., 2008. Computational thinking and thinking about computing. Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences, 366(1881), pp.3717-3725.
[25] Woodfield, R., 2014. Undergraduate retention and attainment across the disciplines. Higher Education Academy, p.45.
[26] Zepke, N., 2013. Student engagement: A complex business supporting the first year experience in tertiary education. International Journal of the First Year in Higher Education, 4(2).