SKILLS

Educating Programmers: A Reflection on Barriers to Deliberate Practice

Michael James Scott & Gheorghita Ghinea

Information Systems, Computing & Mathematics, Brunel University, UK

Corresponding author:
Michael James Scott, Information Systems, Computing & Mathematics, Brunel University, Uxbridge, Middlesex UB8 3PH, UK
Email: michael.scott@brunel.ac.uk, Web: www.p-shift.co.uk

Abstract

Programming is a craft that often demands that learners engage in a significantly high level of individual practice and experimentation in order to acquire basic competencies. However, practice behaviours can be undermined during the early stages of instruction. This is often the result of seemingly trivial misconceptions that, when left unchecked, create cognitive-affective barriers that interact with learners' self-beliefs, potentially inducing emotions that inhibit practice. This paper seeks to ascertain how to design a learning environment that can address this issue. It is proposed that analytical and adaptable approaches, which might include soft scaffolding, ongoing detailed informative feedback and a focus on self-enhancement alongside skill development, can help overcome such barriers.

Keywords: computer science education, computer programming, laboratory instruction, affective development, feedback, self-beliefs, barriers

1. Introduction

Recently, there has been a drive to revitalise computing education (Gove 2012), in part, due to criticisms published by the Nesta Trust (Livingstone & Hope 2011) and the Royal Society (Furber 2012). Unfortunately, few beginners appear to find writing code easy and enjoyable (Jenkins 2001, 2002), so crafting an effective learning environment is not a trivial task. Moreover, despite considerable research into programming instruction since the inception of computer science as an academic discipline, many learners do not acquire the desired level of competency in their first course (Soloway *et al.* 1983, McCracken *et al.* 2001, Tew & Guzdial 2011). Even those who appear to perform well in early tutorials choose not to pursue the discipline (Beaubouef & Mason 2005, Carter 2006). Unfortunately, such issues are so pervasive that the British Computer Society (BCS) declared programming a 'grand challenge' for education research (McGettrick *et al.* 2005).

An important aspect of this challenge is encouraging learners to engage in frequent practice. Evidence suggests that levels of *effort* (Ventura 2005), *comfort* (Wilson & Shrock 2001) and *depth* (Simon *et al.* 2006) predict success in a first programming course. This is

doi:10.11120/stem.hea.2013.0005

in line with the theory that it can take approximately ten years of deliberate practice to become an expert (Ericsson *et al.* 1993, Winslow 1996, Ericsson 2006). Unfortunately, learners often claim that they have “no time” or have “no motivation” to do so (Kinnunen & Malmi 2006, p104). So, if deliberate practice is a key element in the acquisition of programming competencies, how do educators create learning environments that successfully encourage practice?

2. Cognitive-affective barriers and deliberate practice

In order to appreciate how to facilitate frequent practice, the barriers that prevent it should be explored. Programming is markedly distinct from other disciplines because proficiency in other areas does not predict success (Byrne & Lyons 2001, Erdogan *et al.* 2008) and some believe that there are no effective aptitude tests (McGettrick *et al.* 2005, Caspersen *et al.* 2007), assuming that aptitudes for programming even exist (Ericsson *et al.*, 1993, Jenkins 2002). This is because the learning material sometimes demands something very novel of new learners (Huggard 2004), drawing on skills that, at present, are seldom developed prior to programming instruction:

> By means of metaphors and analogies we try to link the new to the old, the novel to the familiar. Under sufficiently slow and gradual change, it works reasonably well; in the case of a sharp discontinuity, however, the method breaks down.
>
> (Dijkstra 1989, p1398)

The sudden sense of ‘radical novelty’ (*ibid.*) constitutes an unexpected challenge for many learners, presenting a barrier to learning. This is because those without prior experience need to adapt to thinking about the intangible and process abstract concepts that are needed to describe the mechanics behind the code they are writing (Du Boulay 1989). Barriers can even arise as early as the first stage of instruction. Consider how someone new to reading program code might conceive the mechanics behind an assignment operation, such as:

```
a = 1;
b = 2;
a = b;          //what is the value of a?
```

Bornat *et al.* (2008) found that for “simple” assignment operations which “hardly look as if they should be hurdles at all” (p54), students held many different mental models for how the program might execute. Even after a few weeks of instruction, some participants failed to apply the correct model consistently in a diagnostic test. This illustrates that the ways in which learners conceptualise computer programs can be diverse, and incorrect models may persist unless there is some intervention. Consequently, it is important not to dismiss the early challenges experienced by individuals as trivial or as constituting a lack of effort or of talent. Put elegantly, “if students struggle to learn something, it follows that this is for some reason difficult to learn” (Jenkins 2002, p53). These issues can be addressed through *soft scaffolding*, such that individual understandings are continuously probed to enable the timely delivery of tailored support (Simons & Klein 2007). Through this, misunderstandings are traced and corrected via the provision of intermediate learning objectives. When not promptly addressed, such issues can impede progress because learners are forced to the edge of, or perhaps beyond, their ‘zone of proximal development’ (Vygotsky 1978, p86).

Yet, Kinnunen & Malmi (2006) note there can be “individual variety in how students respond to the same situation” (p107). Many learners who encounter such challenges are able to overcome them without assistance, albeit perhaps after some frustration. So why are some people tenacious while others seem helpless? A potential candidate for mediating this response is an individual's academic beliefs (Kinnunen & Beth 2012), notably, implicit beliefs surrounding programming aptitude (Murphy & Thomas 2008). Dweck (1999, 2002) divides learners into *entity-theorist*s, who believe their aptitude is a natural fixed trait, and *incremental-theorists*, who believe their aptitude is a malleable quality that is increased through effort. These two groups demonstrate different behaviours when they encounter difficulty (*ibid.*), as summarised in Table 1.

Table 1 Potential influence of different theories of aptitude (adapted from Dweck 2002).

	Entity-Theorists	Incremental-Theorists
Goal of the student	To demonstrate a high coding ability	To improve coding ability, even if poor progress revealed
Meaning of failure	Indicator of low programming aptitude	Indicative of lack of effort, strategy, or prerequisites
Meaning of effort	Demonstrates low programming aptitude	Method of enhancing programming aptitude
Strategy when meets difficulty	Less time practising	More time practising
Performance after difficulty	Impaired	Equal or improved

Too often, it is the case that learners start to believe an inherent aptitude is required to become a programmer. Such beliefs inhibit practice. Thus, it is important that programming pedagogies reinforce the incremental theory. An example might include the liberal use of *detailed informative feedback.* This approach focuses on improvement through illustrating weaknesses to overcome, rather than merely labelling learners with summative grades. The latter might be interpreted as a judgement of aptitude. Furthermore, as many learners "often focus on topics associated with assessment and nothing else" (Gibbs & Simpson 2004, p14) some form of marking is often necessary as an extrinsic motivator. Such marking should be complemented with feedback that helps students understand that programming requires a surprising amount of time and effort, as this has been shown to enhance mindsets when coupled with appropriate instruction on the neuroscience underpinning Dweck's theory (Cutts *et al.* 2010).

While Dweck's (1999, 2002) classification of learners' theories is useful in illustrating some differences, it does not explain why some learners seem far more determined than others. Potential factors, as Huggard (2004) and Rogerson & Scott (2010) affirm, are the negative affective states that learners can experience as they write code. These "states [,] such as frustration and anxiety [, can] impede progress toward learning goals" (McQuiggan *et al.* 2007, p698). However, while some learners become overtly frustrated with the 'all or nothing' nature of preparing a computer program for compilation, others press on without complaint, demonstrating an admirable level of experimentation and debugging proficiency. This can be somewhat surprising given that anything short of a completely syntactically correct set of coded instructions will result in failure, and it is unusual for those at an introductory level to write robust code on their first attempt.

A potential candidate for mediating how learners are able to overcome negative affect is academic self-concept. That is, "self-perceptions formed through experience with and interpretations of one's environment" (Marsh & Martin 2011, p60). Many domain-specific forms of self-concept, such as programming self-concept, demonstrate a reciprocal

Proc. HEA STEM Conf. (2013)
doi:10.11120/stem.hea.2013.0005

relationship with academic achievement in their respective area (*ibid.*) as well as, more generally, interactions with study-related emotions (Goetz *et al.* 2010). Extending this notion, learners who *believe* that they are programmers, those with a high programming self-concept, may be able to overcome frustrations and anxiety more easily, thereby maintaining high levels of motivation. So, how can self-concept be enhanced? A meta-analysis of 200 interventions shows that practices which target a domain-specific facet of self-concept, with an emphasis on motivational praise and feedback alongside skill development, yield the largest effects (O'Mara *et al.* 2006). Other aspects of effective practice might also emphasise learning activities that are enjoyable and nurture a sense of pride (Goetz *et al.* 2010).

3. Conclusion

Learners often need to practice writing code frequently in order to acquire basic programming competencies. This paper questions how learning environments can be better designed in order to facilitate deliberate practice, describing three potential barriers to such practice: the radical novelty of the learning material; the belief that some inherent aptitude is required; and the emergence of unfavourable affective states. It is proposed that examples of good practice might include: soft scaffolding; on-going informative feedback that encourages a growth mindset; and an emphasis on self-enhancement through motivational feedback and pride-worthy activities in addition to skills development. However, empirical research is needed to establish the potential impact of these problems and proposals.

References

Beaubouef, T. and Mason, J. (2005) Why the high attrition rate for computer science students: some thoughts and observations. *ACM SIGCSE Bulletin* **37** (2), 103–106.

Bornat, R., Dehnadi, S. and Simon, S. (2008) Mental models, consistency and programming aptitude. In *Proceedings of the 10th Australasian Computing Education Conference*, pp53–61.

Byrne, P. and Lyons, G. (2001) The effect of student attributes on success in programming. *ACM SIGCSE Bulletin* **33** (3), 49–52.

Carter, L. (2006) Why students with an apparent aptitude for computer science don't choose to major in computer science. *ACM SIGCSE Bulletin* **38** (1), 27–31.

Caspersen, M., Benedsen, J. and Larsen, K. (2007) Mental models and programming aptitude. *ACM SIGCSE Bulletin* **39** (3), 206–210.

Cutts, Q., Cutts, E., Draper, S., O'Donnell, P., and Saffrey, P. (2010) Manipulating Mindset to Positively Influence Introductory Programming Performance. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* pp431–435.

Dijkstra, E.W. (1989) A debate on teaching computer science: on the cruelty of really teaching computer science. *Communications of the ACM* **32** (12), 1398–1404.

Du Boulay, J. (1989) Some difficulties of learning to program. *Educational Computing Research* **2** (1), 57–53.

Dweck, C.S. (1999) *Self-Theories: Their Role in Motivation, Personality, and Development*. Philadelphia, PA, USA: Psychology Press.

Proc. HEA STEM Conf. (2013)
doi:10.11120/stem.hea.2013.0005

Dweck, C. (2002) Messages that motivate: how praise molds students' beliefs, motivation, and performance (in surprising ways). In *Improving Academic Achievement* (ed. J. Aronson). San Diego: Academic Press, pp37–60.

Erdogan, Y., Aydin, E. and Kabaca, T. (2008) Exploring the psychological predictors of programming achievement. *Instructional Psychology* **35** (3), 264–270.

Ericsson, K. (2006) The influence of experience and deliberate practice on the development of superior expert performance. In *The Cambridge Handbook of Expertise and Expert Performance* (ed. K. Ericsson *et al.*). Cambridge: Cambridge University Press, pp683–703.

Ericsson, K., Krampe, R. and Tesch-Romer, C. (1993) The role of deliberate practice in the acquisition of expert performance. *Psychological Review* **100** (3), 363–406.

Furber, S. (2012) *Shut Down or Restart? The Way Forward for Computing in UK Schools*. London: Royal Society.

Gibbs, G. and Simpson, C. (2004) Conditions under which assessment supports students' learning. *Learning and Teaching in Higher Education* **1**, 3–31.

Goetz, T., Cronjaeger, H., Frenzel, A., Lüdtke, O. and Hall, N. (2010) Academic self-concept and emotion relations: domain specificity and age effects. *Contemporary Educational Psychology* **35** (1), 44–58.

Gove, M. (2012) Digital Literacy and the Future of ICT in Schools. Presentation at the BETT Show, Department for Education. Available at http://www.education.gov.uk/inthenews/speeches/a00201868/michael-gove-speech-at-the-bett-show-2012 (accessed 5 November 2012).

Huggard, M. (2004) *Programming trauma: can it be avoided?* In *Proceedings of the BCS Grand Challenges in Computing: Education*, pp50–51.

Jenkins, T. (2001) Teaching programming: a journey from teacher to motivator. In *Proceedings of the 2nd HEA Conference for the ICS Learning and Teaching Support Network*, pp65–71.

Jenkins, T. (2002) On the difficulty of learning to program. In *Proceedings of the 3rd HEA Conference for the ICS Learning and Teaching Support Network*, pp53–58.

Kinnunen, P. and Malmi, L. (2006) Why students drop out CS1 courses? In *Proceedings of the 2nd International Computing Education Research Workshop*, pp97–108.

Kinnunen, P. and Beth, S. (2012) My Program is OK - Am I? Computing Freshman's Experience of Doing Programming Assignments. *Computer Science Education* **22** (1), 1–28.

Livingstone, I. and Hope, A. (2011) *Next Gen: Transforming the UK into the World's Leading Talent Hub for the Video Games and Visual Effects Industries*. London: NESTA.

Marsh, H. and Martin, A. (2011) Academic self-concept and academic achievement: relations and causal ordering. *British Journal of Educational Psychology* **81** (1), 59–77.

McCracken, M., Almstrum, D., Diaz, M., Guzdial, D., Hagen, Y., Kolikant, C., Laxer, L., Thoman, I., Utting, T., and Wilusz, T. (2001) A multi-national, multi-institutional study of assessment of programming skills of first year CS students. *ACM SIGCSE Bulletin* **33** (4), 125–140.

McGettrick, A., Boyle, R., Ibbett, R., Lloyd, J., Lovegrove, G. and Mander, K. (2005) Grand challenges in computing: education: a summary. *The Computer Journal* **48** (1), 42–48.

McQuiggan, S., Lee, S. and Lester, J. (2007) Early prediction of student frustration. *Affective Computing and Intelligent Interaction* **47** (38), 698–709.

Murphy, L. and Thomas, L. (2008) Dangers of a Fixed Mindset: Implications of Self-Theories Research for Computer Science Education, *ACM SIGCSE Bulletin* **40** (3), 271–275.

Proc. HEA STEM Conf. (2013)
doi:10.11120/stem.hea.2013.0005

O'Mara, A.J., Marsh, H.W., Craven, R.G. and Debus, R.L. (2006) Do self-concept interventions make a difference? A synergistic blend of construct validation and meta-analysis. *Educational Psychologist* **41**, 181–206.

Rogerson, C. and Scott, E. (2010) The fear factor: how it affects students learning to program in a tertiary environment. *Information Technology Education* **9** (1), 147–171.

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 *Proceedings of the 8th Australasian Computing Education Conference*, pp189–196.

Simons, K. and Klein, J. (2007) The impact of scaffolding and student achievement levels in a problem-based learning environment. *Instructional Science* **35**, 41–72.

Soloway, E., Bonar, J. and Ehrlich, K. (1983) Cognitive strategies and looping constructs: an empirical study. *Communications of the ACM* **26** (11), 853–860.

Tew, E. and Guzdial, M. (2011) The FCS1: A language independent assessment of CS1 knowledge. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, pp111–116.

Ventura, P. (2005) Identifying predictors of success for an objects-first CS1. *Computer Science Education* **15** (3), 223–243.

Vygotsky, L. (1978) *Mind in Society: The Development of Higher Psychological Processes*. London: University Press.

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

Winslow, L.E. (1996) Programming pedagogy – A psychological overview. *ACM SIGCSE Bulletin* **28** (1), 17–22.

doi:10.11120/stem.hea.2013.0005