**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...
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-theorists, 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).

<table>
<thead>
<tr>
<th>Goal of the student</th>
<th>Entity-Theorists</th>
<th>Incremental-Theorists</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meaning of failure</td>
<td>Indicator of low programming aptitude</td>
<td>Indicative of lack of effort, strategy, or prerequisites</td>
</tr>
<tr>
<td>Meaning of effort</td>
<td>Demonstrates low programming aptitude</td>
<td>Method of enhancing programming aptitude</td>
</tr>
<tr>
<td>Strategy when meets difficulty</td>
<td>Less time practising</td>
<td>More time practising</td>
</tr>
<tr>
<td>Performance after difficulty</td>
<td>Impaired</td>
<td>Equal or improved</td>
</tr>
</tbody>
</table>

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
relationship with academic achievement in their respective area \cite{ibid} as well as, more generally, interactions with study-related emotions \cite{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 \cite{O'Mara et al. 2006}. Other aspects of effective practice might also emphasise learning activities that are enjoyable and nurture a sense of pride \cite{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


