Why the bottom 10% just can't do it -
Mental Effort Measures and Implication for Introductory Programming Courses

Raina Mason¹ Graham Cooper¹

¹ Southern Cross Business School
Southern Cross University
Hogbin Drive, Coffs Harbour, New South Wales 2450
raina.mason@scu.edu.au
graham.cooper@scu.edu.au

Abstract
This paper reports the results of mental effort measures and comments collected as part of a study of 44 introductory programming courses in 28 Australian universities, conducted in the latter months of 2010. Academic staff were interviewed regarding their perceptions of the mental effort that is required by themselves, an average student, and a low-performance student while attempting to solve and learn from a novice programming problem.

Qualitative responses were also gathered from academics to gain insight into the various student profiles and impediments to learning for low-performing students. Mental effort results indicated that many low-performance students typically experience high to extreme levels of mental effort. Verbal responses obtained from academics also indicate an awareness that for many low-performance students learning fails due to excessive demands being placed upon their cognitive resources.

It is suggested that for many low-performance students learning fails due to cognitive overload. The implications for the selection of languages and environments and for the design of introductory programming courses (units) are discussed.

Keywords: introductory programming, programming success, pedagogy, Australian university courses, mental effort measures, cognitive load, cognition.

1 Introduction
A survey of introductory programming courses (units) offered in Australian universities was conducted at the end of 2010. This was designed to repeat similar studies regarding introductory programming courses within Australian universities that had been conducted in 2001 (de Raadt, Watson & Toleman, 2002) and 2003 (de Raadt, Watson & Toleman, 2004). The findings of the 2010 survey are reported in Mason et al. (In press).

The primary focus of the 2010 survey was to identify the programming language(s) used within Australian university courses, along with teaching-learning support factors such as development environment, textbook used (if any), hours of teaching and focus upon problem solving. Differences between the 2001/2003 censuses and the 2010 study are suggestive of trends in approaches to teaching introductory programming within Australian universities. The major changes were shifts towards some languages (Python and C#) at the expense of other languages (Visual Basic and C++). There was also an increased usage of IDEs and other tools. Discussion was also provided regarding perceived reasons offered by interviewees (academics teaching introductory programming courses) for such changes, which were predominately reported to be pedagogical.

A secondary focus of the survey was to investigate the evaluations by introductory programming academics of the level of “mental effort” that would be required by students studying in these programming courses. The primary outcome of the study in this regard is that academic staff instructing in introductory programming courses rated their own levels of mental effort in dealing with the course (unit) content to be “low”, the mental effort of average students to be “above average”, and the mental effort of students in “the bottom 10% of performance levels” to be “very high to extreme”.

The current paper further explores the role of mental effort within the cohort of “the bottom 10% of performance”.

2 The Role of Mental Effort in Learning

2.1 Mental Effort
Mental effort is a cognitive construct that has its derivation in Cognitive Load Theory (Sweller 1999) and refers to the deliberate focus of attention and level of cognitive resources that are allocated to a task (Kirschner 2002).

Mental effort is closely related to cognitive load. Cognitive load is a “count” of the number of elements that must be held within working memory for any particular processing task (Sweller 1999). Each element will be in the form of a schema (Chi, Glaser & Rees 1982) as it exists for an individual based upon their current personalised knowledge base. Schemas are well organised hierarchical networks of conceptual and procedural information. As ones’ expertise in a content domain expands, organises and intercepts, so too does ones’ schemas become larger, more well organised and more interrelated. A novice in a content domain only holds small disconnected schemas, and so needs to hold a larger number of elements in working memory, than does...
an “expert” in the area, when consciously attending to the same body of content.

“Mental effort” is a term considered to be more accessible to lay persons due to its alignment to a general enquiry regarding “how hard one is thinking”, rather than a more load-orientated enquiry of “how many things one is thinking about”. Mental effort has historically been used as a measure to evaluate cognitive load (Paas & Van Merrienboer 1993).

2.2 The Targets of Mental Effort

Not all content is equally challenging to learn. There are three distinct factors which may contribute to a student’s experienced ease or difficulty in learning a body of to-be-learned content. These may be identified as intrinsic, extraneous and germane factors.

Some content, by its intrinsic nature, is considered to be “difficult”, and thus requires a high level of mental effort to comprehend and learn. This is primarily due to the inherent complexities, interrelations and subtleties of the information. Both mathematics and computer programming are such intrinsically difficult areas (Sweller & Chandler 1994).

In the context of learning computer programming there are at least five distinct domains that must be mastered at the same time. These have been identified by du Boulay (1989) as: general orientation, the notional machine, notation (syntax and semantics), structures, and pragmatics (developing, debugging, etc). The current paper argues that this multi-domain aspect of programming is a primary contributor to its reputation as being difficult to learn.

Another potential source of difficulty in learning content lies in the way in which to-be-learnt information is presented. This is extraneous to the conceptual understanding of content, yet needs to be processed by a learner’s cognitive resources because it is the means by which information is presented to the learner. On this basis it may be the “materials”, or “media”, or “activities” involved in the teaching-learning transaction (Ayres & Sweller 2005).

For example, text based instructions specifying how to travel from one location to another are usually more difficult to understand than an equivalent-content source based around the use of visual maps. This is true regardless of whether one is travelling from home to a friend’s house, or from one city to another. Maps have an obvious spatial relation to the 2-dimensional landscape which they represent, while the textual form does not.

Similarly, different programming languages and environments may present differing levels of visual representations of core programming concepts, such as loops (iterations). The choice of computer programming language and/or environment may have a direct impact upon the ease of comprehending, learning and applying underlying programming concepts. The language itself is extraneous to understanding these core programming concepts, yet it is the primary vehicle by which these are usually presented to a student. A student who does not hold schemas for core programming concepts and who falters on aspects of language or syntax, will likely be unable to distinguish between such core concepts versus aspects of the language, as to why he or she has faltered.

The third factor that gives rise to mental effort is from the conscious focus of attention that a learner brings to a task to deliberately “remember” and/or “understand” to-be-learnt content. In addition to this, a learner may also seek to proceduralise this newly acquired knowledge base through strategies such as drill-and-practice and mental rehearsal (Cooper, Tindall-Ford, Chandler & Sweller 2001). In this context mental effort is germane to the tasks of schema acquisition and automation (Paas & Van Merrienboer 1994). These activities generally consist of making mental comparisons and contrasts between one’s already-held knowledge base (their schemas) and the newly presented information, along with some form of practice to proceduralise the application of these evolving schemas. A broad instructional strategy that benefits learning is to maximise the level of cognitive resources dedicated to such germane activities (Paas, Renki & Sweller 2003).

Depending upon the nature of the to-be-learnt content germane activities may range from shallow processing tasks such as rote-learning to deep processing elaboration strategies. For example, consider a student studying chemistry. The elements of the periodic table may be rote-learnt as a simple sequence, or more deeply processed and linked to other facts that may already be held, or developing, such as atomic weight, valency and chemical reactions.

Similarly, programming may be viewed as a set of isolated base concepts to be rote-learnt, or analysed, abstracted and re-organised into a wide range of algorithms that are utilised within programming structures. In general, the deeper the level of processing, the more able will the learner be in recalling and applying the information due to the level of “understanding” that has taken place, rather than only memorisation.

The bottom line, from a Cognitive Load Theory perspective (Paas, Renki & Sweller 2004), is that these three demands of mental effort compete for cognitive resources. An issue arises because human cognitive resources are strictly limited (Miller 1956) and so the combined demands of these three targets of mental effort cannot always be met. The reality is that students, in attempting to deal with the broad complexity of programming, in conjunction with a specific programming language and/or environment, may have insufficient cognitive resources left available to dedicate to the primary purpose of the teaching and learning transaction - that of actually engaging in germane tasks such as comprehending, learning and proceduralising the newly presented to-be-learnt content regarding core programming concepts.

It is important to note that these potential difficulties described in comprehending and learning may be greatly amplified for students at the lower end of the performance spectrum.

The next section briefly outlines some of the key features of human cognitive architecture and Cognitive Load Theory, before returning to the findings of the 2010 survey, and the inclusion of questions regarding mental effort.
Human Cognitive Architecture is often described as an information processing model with operational relations between the separate conceptual functions of Long Term Memory and Working Memory (Sweller 1999). There are added complexities to the model with the inclusion of Sensory Memory to handle, very briefly, the input from our senses, and modality specific slave-systems for management and integration of distinct visual and auditory information components within Working Memory (Mayer 2005). While these have significant implications for the processing of instructional materials, they will be left largely unexplored in the current paper.

Long term memory is a virtually permanent store of knowledge. It is effectively unlimited in its capacity, with knowledge stored in the form of “schemas” which are hierarchically structured information networks that gather and interrelate throughout a person’s lifetime (Sweller 1999).

In contrast, Working Memory is dedicated to the conscious processing of information. It is where activities such as thinking and problem solving take place. The elements of knowledge, which are held in the Long Term Memory store as schemas, are utilised within working memory processing tasks (Sweller 1999).

Working memory is strictly bounded in its capacity with 7 elements (+/-2) representing typical performance by adults on random memory tasks (Miller 1956). This number will be further reduced when attending to elements that are not random, but hold well-defined relations. For example, the formula “F=ma” does not only require holding of the concepts of force, mass and acceleration, but also their relation as defined by the formula.

The limitations of Working Memory may be worked-around by humans through selecting “larger” size elements to process. These elements are the schemas that one already holds in Long Term Memory. For example, the schema for “alphabet” can easily be decoded into a set sequence of 26 letters that most readers will well know.

A second primary performance aspect of human cognitive processing models is “automation”, which refers to a person’s ability to proceduralise and apply their schemas while needing only very low levels of conscious attention (Shiffrin & Schneider 1977).

Models of expertise typically reduce the primary traits of expertise to the possession of relatively large well-structured networks of schemas, and the capacity to access and apply the knowledge and skills held in these schemas, with low levels of conscious attention. These two attributes are sufficient to explain virtually all aspects of expert performance, including transfer and problem solving (Cooper & Sweller 1987).

Cognitive Load Theory (Paas, Renki & Sweller 2004) explores the application of information processing models to the processes of learning, and more importantly, to the design of instructional materials and activities, with the purpose of facilitating learning. Central to this approach of instructional design is the limitations of working memory. If the mental processing load placed upon these limited resources are exceeded at any point during a learning transaction (“overloaded”), then learning will falter due to the dropping of information that was currently being processed.

There are many apparent similarities between this model of human cognitive architecture and its manner of information processing, with those associated with computer architectures and data processing. This is not accidental. There have long been identified relations between these two areas that have effectively evolved in parallel (Hunt 1982). If you (the reader) have a high knowledge base in computer programming and architecture, but a relatively low knowledge base in human cognitive architecture, then you may think of “cognitive overload” as being akin to a “buffer overflow”. Overloading means that some data is dropped. Once dropped, it is lost irretrievably - and so while some form of processing may still occur, the nature of the information now being handled is incomplete, or worse still, incorrect and can result in an effective program crash, or memory access errors.

### 3.2 Sources of Cognitive Load

There are three identified sources of cognitive load. These are Intrinsic, Extraneous and germane (Paas, Renki & Sweller 2003). Each was described in lay form earlier in this paper in the context of mental effort. Each is dealt with again from a more technical perspective.

#### 3.2.1 Intrinsic Cognitive Load

Intrinsic cognitive load refers to the innate difficulty of a body of to-be-learned content. Information that contains many internal relationships between knowledge elements (element interactivity) imposes a higher level of cognitive load than information that can be considered to be a series of simple, unrelated facts (Sweller & Chandler 1994). This is because it is not only the separate discrete elements that need to be consciously attended to, but also the relations between those elements. A popular example to consider is the learning of a second language -- not a second programming language, but a second spoken language. Assume that you, the reader, are relatively expert at English, and learn Indonesian, French or Spanish. Each is written in a similar script to English and has a large level of overlap in phonemes.

In learning a second language (or a third, fourth and so on), there at least two distinct aspects that need to be learned. These are the vocabulary of the language and the syntax (or grammar) of the language. The vocabulary is relatively simple, being in general terms a one-to-one word-swap for “me”, “you”, “good”, “bad”, “morning”, “evening” and so on. There are several hundred, perhaps thousand, such word-swaps required in order to be able to communicate meaningfully at a conversational level. The second aspect, the syntax, if far more difficult to learn, because to construct a sentence such as “How are you this morning?” is generally not just a process of swapping words. All of the words may need to be considered simultaneously, along with their inter-relations. Note, too, that this is a simple, common sentence! If there are more subtle aspects being included such as personal
relationship, respect associated with maturity (age of the persons involved), time of day being said, and the gender of the items (not the people, but the items such as table, chair, food) in question (of which there is no meaningful equivalent in English) then the translation can become much more difficult if there is a requirement to “be grammatically correct”.

A question such as “Have you eaten dinner this evening?” needs much more than a simple word-swap process to be translated into many languages because the order of the words may need to be re-sequenced, there are likely issues of verb conjugation, there is a need to be mindful of the “respect-relationship” between the players (akin to using “sir”, “Mr” or “mate” in the Australian context), and we have not even yet considered aspects of enunciation and emphasis.

The purpose of this discussion of language is to indicate that learning a second language contains a mix of intrinsically relatively “easy” content - the one-to one word swaps - plus the relatively “difficult” content of syntax, which is difficult due to the element interactivity between the separate words. On top of this there may be physical performance difficulties such as recognising and producing specific sounds (phonemes). We will return to the issue of learning a language later in this paper, although the context will then move to that of computer programming languages.

3.2.2 ExTRANeous Cognitive Load

ExTRANeous cognitive load refers to the load generated by the format of instructional materials and/or to the performance of learning activities. Some formats and activities hinder learning by loading the learner with unnecessary information processing (unnecessary to the task of acquiring schemas and automating them). ExTRANeous cognitive load is open to being manipulated by the instructional designer. There are several common effects that have been investigated by Cognitive Load Theory. Two that are relevant for discussion here in the context of introductory programming are the Worked-Example Effect (Cooper & Sweller 1987) and the Split-Attention Effect (Chandler & Sweller 1992).

Many studies have indicated that a relatively high weighting upon problem solving tasks is less effective at facilitating learning than studying a set of similar worked examples (Renki 2005). The underlying theoretical analysis is based upon the argument that novices use means-ends-analysis to solve problems, that this process imposes a relatively high level of cognitive load, directs attention to the answer, rather than the process of obtaining the answer, and consequently impedes learning (Sweller 1988). It was observed though the survey that many introductory programming courses still place a relatively high emphasis upon problem solving. By this we mean that a core, central activity within the learning materials, is to require students to be actively engaged in producing a solution to a computer programming problem solving task. Such a focus on problem solving may hinder learning.

The Split Attention Effect (Chandler & Sweller 1992) argues that split formats of instruction, whereby two or more mutually referring sources of instruction are presented in isolation, and thus need to be mentally integrated to enable comprehension, will be less effective at facilitating learning than an equivalent-content integrated format where all mutually referring sources of information have been integrated into a single source.

For example, consider a split attention format that may result from a physical layout of instructional materials that “splits” the location of instructional text from the location of a text-referenced diagram which usually sits beside or below the text. This will impose a level of visual search and require a mental integration of meaning between the textual and graphical components of instruction to enable comprehension. This search and mental integration is not required for an equivalent-content presentation where the textual information has been integrated (embedded) within the diagram. The added cognitive load due to the visual search and mental integration of the two mutually referring sources of information in a split format causes additional taxing of cognitive resources, and so reduces the cognitive resources available to be applied to the germane activities associated with schema acquisition and automation. As a direct result of such an instructional design, learning is reduced. This effect has been demonstrated to be present in many instructional settings and is highly robust (Sweller 1999).

We will return later to the split attention effect, not to discuss the physical integration of materials, but to discuss the split-attention between learning underlying concepts of computer programming compared to the learning of a specific computer programming language (along with syntax). A common feature of introductory programming courses appears to be that the core programming concepts are “cocooned” within a specific computer programming language and syntax.

3.2.3 GerMANe Cognitive Load

Germane cognitive load refers to load devoted to the processing, construction and automation of schemas. This application of cognitive load is beneficial to learning. A general strategy in many instructional settings is to reduce extraneous load, and to direct the released cognitive resources towards the germane efforts associated with schema acquisition and automation (Paas, Renki & Sweller 2004).

In situations where both intrinsic and extraneous loads are high it is likely that this will effectively block the capacity for germane load, and thus block learning.

It is argued in the current paper that this is the situation experienced by many low performing students who are undertaking introductory programming courses. These students, who lack schemas in programming, are faced with the intrinsic complexity of core programming concepts embedded within an instructional presentation that involves aspects of computer language and syntax which are extraneous to this task. It is likely that these students will have insufficient cognitive resources available to enable the process of learning.

4 Cognitive Load and Novice Programming Problems

The current survey explored aspects of cognitive load associated with the teaching and learning of programming
concepts within introductory programming courses offered by Australian universities.

Specifically, in the context of learning the generic concepts of programming through completing computer programming problem solving tasks, there are three separate components of processing that need to be attended to, and it is argued that these broadly align to the three different sources of cognitive load. These are:

- **Intrinsic load**, associated with the concepts and interpretation of problem statements;
- **Extraneous load**, determined by the language and environment, along with associated constraints such as syntax; and
- **Germane load**, associated with the cognitive processing to acquire and automate new schemas.

The term “mental effort” has been used to communicate with participants rather than “cognitive load”. This self-reported assessment of perceived mental experience is taken as a measure of the cognitive load associated with a task performance.

Participants in the survey were asked to rate their own levels of mental effort on each of the three components of cognitive load, using a 9 point Likert scale (where 1 = ‘no mental effort’ and 9 = “extreme mental effort”). Participants were also asked to estimate the levels of mental effort on each of these components experienced by an average student in their introductory programming course and that experienced by a student in the ‘bottom 10% of performance’ in their course.

There were some participants who did not immediately offer a response for all, or an aspect, of this series of questions, particularly for the “bottom 10% of performance”. These were removed from the first series of analyses, but will be commented upon further below.

An initial analysis was performed on the mental effort scores given by participants (instructors), for each of these three components of cognition -- understanding the problem statement, using the environment, and reinforcing previous concepts. A series of comparisons between, firstly, the self-rating of the participant (a first year programming instructor) and the rating anticipated to be experienced by an ‘average student’; and then between the anticipated average student’s rating and the anticipated level to be experienced by a student in the bottom 10% of the class’s performance. The comparisons were performed using a series of Wilcoxon Signed Rank tests.

These results were reported in Mason et al. (In press), and are replicated here to enable further analysis and discussion. Table 1 shows the mean, median and mode for each of these cognitive load areas for instructors, the average student and students “in the bottom 10%”. The data was heavily skewed, so measures of central tendency provided are modes and medians.

Table 2 shows the results of the Wilcoxon Signed-rank tests (one-tailed) for within-subject comparisons between instructors and average students, and between average students and students in the bottom 10%.

### Table 1: Levels of mental effort reported

<table>
<thead>
<tr>
<th>Instructor</th>
<th>Average Student</th>
<th>Bottom 10% Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>median</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

### Table 2: Wilcoxon Signed-Rank Tests

<table>
<thead>
<tr>
<th></th>
<th>W</th>
<th>Ns/r</th>
<th>z</th>
<th>p</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding and processing the problem statement</td>
<td>811</td>
<td>43</td>
<td>4.39</td>
<td>&lt;0.0001</td>
<td>43</td>
</tr>
<tr>
<td>Navigating/using the environment, tools or language</td>
<td>838</td>
<td>41</td>
<td>5.43</td>
<td>&lt;0.0001</td>
<td>43</td>
</tr>
<tr>
<td>Learning from the problem/ reinforcing previous concepts</td>
<td>647</td>
<td>40</td>
<td>4.34</td>
<td>&lt;0.0001</td>
<td>42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>W</th>
<th>Ns/r</th>
<th>z</th>
<th>p</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding and processing the problem statement</td>
<td>207</td>
<td>24</td>
<td>2.95</td>
<td>0.0016</td>
<td>25</td>
</tr>
<tr>
<td>Navigating/using the environment, tools or language</td>
<td>276</td>
<td>23</td>
<td>4.19</td>
<td>&lt;0.0001</td>
<td>25</td>
</tr>
<tr>
<td>Learning from the problem/ reinforcing previous concepts</td>
<td>144</td>
<td>20</td>
<td>2.68</td>
<td>0.0037</td>
<td>21</td>
</tr>
</tbody>
</table>

These results indicate that for each of the three sources of cognitive load, the instructors in the introductory programming courses rated their own levels of required mental effort to be low, and that they expected that average students would need to exert higher levels of mental effort than themselves - ‘above average’.

Additionally, again for each of the three sources of cognitive load, the participants rated the anticipated mental effort to be experienced by a student in “the bottom 10%” to be higher than that of an average student, rating at very high to extreme mental effort.

These results are consistent with the argument provided earlier regarding cognitive overload. Students who are in the bottom 10% of performance are perceived by the introductory programming instructors to be effectively swamped in mental effort on each of the three measures. As these three areas compete for cognitive resources it is unlikely that students can allocate such high levels to all three components, despite their desires to do so, and despite the instructors perceptions that students are attempting to do so.

Human cognitive processes are limited. Student have no option but to try and understand and respond to the programming problem statement, as this is the primary task to which they have been explicitly directed to engage with. The student also has no option but to navigate, as best he or she can, the programming language and syntax,
as this is the environment which they have explicitly been directed to operate within. Given that these two tasks are each placing very heavy processing burdens upon mental resources it may be expected that the third area, those of germane activities, are effectively strangled for processing resources.

Recall that not all participants gave a score for the mental effort measures, instead choosing to provide comment. The second layer of analysis, now pursued, focuses upon the verbal comments provided by all participants regarding their perceptions of mental effort for the different student cohorts.

4.1 Participants that did not give a score for all or any of the cognitive load aspects

A total of 44 participants took part in the survey, and all of these participants answered the questions regarding the mental effort experienced by themselves on each of the three factors while solving a novice programming problem. One of the participants declined to answer the questions regarding average and ‘bottom 10%’ students as he/she indicated that he/she believed “that the ‘average student’ could be designated as any of the scores on the Likert scale” We consider this to indicate a misinterpretation of our line of enquiry, and no further consideration is given to the responses provided by this participant. Of the 43 remaining participants, one declined to give a score for the “germane” aspect for the participant. Of the 43 remaining participants, one declined to give a score for the “germane” aspect for the “average student”, whilst offering the scores for the other aspects.

Only 25 participants gave any scores for the “bottom 10%” and often this was coupled with qualifying statements. Only 21 of these participants were willing to specify a score for mental effort for the germane aspect for students in this bottom 10% of performance in the course.

5 Participants’ Comment Analysis

Many participants indicated that providing an estimate of mental effort for lower performing students was a much more complex scenario than could be expressed by a simple numerical answer to a Likert scale question. Even those who did give a score often added comments to qualify their answer.

5.1 Methodology of further analysis

The phone interviews with the 44 participants in the survey were audio-recorded with their permission, with the exception of one participant, who agreed to be surveyed but did not give permission to be recorded. For this particular interview, comprehensive notes were taken and recorded by hand on the interview script. The remaining audio-recordings were then transcribed.

Truncated versions of each interview transcript were created for the purposes of further analysis, temporarily disregarding data on languages, textbooks, on-campus hours and other aspects of introductory programming courses within Australian universities, which have been reported elsewhere (Mason et al. In press). Data retained included mental effort measures for “students in the bottom 10% of the course” (where given), any transcribed comments that offered insight into mental effort (or lack of such) used by students, and any other comments that were offered by participants throughout the course of the interview that had bearing on possible reasons for success or failure of students in the bottom 10% of the course.

Interviews were confidential and participants were encouraged to be open, frank and fearless. On this basis the researchers have redacted phrases that may be considered to be disrespectful of students as clients of the education system. Thematic analysis of the comments has been performed, but in some cases specific terminology has been withheld. In these cases, “[redacted]” has been used. This indicates terminology (not profanity) that may be interpreted as offensive to the people thus identified and labelled.

5.2 Student Profiles

This survey asked for Likert scale measures for mental effort for students in the bottom 10% of course performance. If the instructor asked the interviewer for clarification of type of student targeted, then the interviewer (using the terminology used by the instructor) indicated that the mental effort expended by a less capable student who is trying was required.

[I]nstructor] “by effort do you mean students that apply themselves that are [redacted]? Or students that don’t apply themselves?”

[In]terviewer] “apply themselves that are [redacted]”

However, some instructors offered two values (one for students who did not try, and those who were trying), and others stated that one simple measure couldn’t be given and instead offered comments. For this reason, the comments associated with all of the types of student profiles, as well as mental effort measures for the student profile of “students who are less capable but are trying”, have been analysed.

The interview summaries were examined for commonalities between participants regarding the profiles of students composing the bottom 10% of their course. Responses indicated that 34% (15/44) of participants explicitly indicated that more than one student profile existed in the student composition of the bottom 10% of their course. All participant summaries were examined to identify the range of profiles of these students.

The profiles of students that were identified by instructors were:

• ‘Strivers’ - less capable students who are actually trying - identified by 75% of participants;
• ‘Idlers’ - those students who attend class but who do not try - identified by 40% of participants;
• ‘Ghosts’ - students who do not attend class are not seen by instructors - identified by 14% of participants.

Each profile type was then examined separately using thematic analysis across participants to determine possible reasons given by participants for lack of success by these students.

5.2.1 Ghosts - students who are not there

A relatively small number of participants (14%) identified that some or all of the students in the bottom 10% of the course were never seen by the instructor, or
5.2.2 Idlers - students who do not apply themselves

Instructors identified two separate profiles of students who did attend classes or were ‘seen’, within the bottom 10% of students. A cohort of “students who attend class but don’t try” were identified by 40% of participants. We have designated this profile of students as “Idlers”. “Strivers” - the remaining profile of students - are examined in the next section of this paper.

Some of the reasons offered by instructors for the Idlers’ lack of progress were similar to those offered for the Ghosts.

Students’ lack of motivation and desire to learn was given as a reason in many cases:

• “... there is very little mental effort because they don’t do anything. In workshops they are on Facebook or chatting or getting the answers from their mates rather than working it out for themselves.” [5]
• “Don’t try. Hand in other people’s assignments, don’t attend, don’t do tasks, are on facebook.” [23]
• “not motivated enough to try” [26]

Several instructors offered the reason for this lack of effort to be that some students were forced to do the unit by their degree requirements, had no intent of going further, did not enjoy programming and that these students didn’t apply themselves to learning:

• “Only about 20% of students [in the course] intend to progress with programming” [23]

5.2.3 Strivers - students who are less capable but are trying

Most (75%) instructors indicated that the bottom 10% of students in their course contained students who were less capable but were trying to succeed. We have designated these students “Strivers”. Recall from Table 1 that the mode on each of the three mental effort measures for the students in the bottom 10% was reported as “9”, the maximum value available on the Likert scales.

Instructors also often offered indicative comments about the level of mental effort for Strivers, regardless of whether they gave a Likert scale number. These comments are offered below to show the general agreement that for this profile of students the mental effort required for all three aspects was very high to extreme, and in some cases “off the scale”. These include:

• “off the scale” [11]
• “putting all their effort in” [15]
• “try very very hard” [16]
• “expend more energy for borderline pass - more effort in getting that borderline pass than others use to get higher marks” [28]
• “very high level of mental effort” [30]
• “factor of 10 at the very minimum” [34]
• “try but can’t solve the problem at all” [35]
• “a lot more effort” [41]

Given that these Strivers are often being reported as trying as hard as they can, it is important to consider the reasons given for why they find it so difficult, or fail (if any reasons were given).

One group of instructors offered the opinion that students had an innate aptitude (or lack of aptitude) for
programming and that if this was missing, those students would never understand programming - no matter how hard they tried. These comments are presented below:

- “some of them pick it up quickly, some take longer and some never pick it up at all.” [7]
- “they struggle - they genuinely try and they put a lot of work in but it’s not how their brains are wired” [22]
- “There are those that get it … and some people for some reason just don’t get it and they are hopelessly lost, and they just never seem to be able to get it at all.” [23]
- “some students have some sort of mental block as far as programming is concerned. They might find it difficult to even follow a set of sequenced step-by-step instructions.” [27]
- “they try really hard but the penny doesn’t drop” [22]

Other instructors commented that the Strivers lacked literacy, comprehension and analysis skills prior to entering the introductory programming course, and that this compounded the difficulty of learning the material for this cohort:

- “they have very poor literacy and comprehension skills” [15]
- “[Programming] is really a bit harder than the other units - it requires some analysis and some mathematical nous almost. And they simply don’t have it.” [34]

Strivers from a non-English-speaking background had a particularly hard time, according to some participants. Some of these students may be more capable (than their programming performance indicates) but the additional cognitive load imposed by working in English as a second language means that they fail to learn or perform:

- “high level of overseas students - sometimes what seems plain to us is not plain to them” [27]
- “often their problem may be that they are not so fluent in English” [41]
- “Non-English students struggle with dealing with the help systems. They are familiar with synonyms so unless the word is exactly the same as the one they have typed in they cannot find it.” [43]

Often instructors offered comments about the difficulties, confusion and extreme mental effort experienced by the Strivers, but did not offer a reason that this was occurring:

- “Extremely difficult for the bottom ones to understand. It’s hard for them. The bottom ones avoid it [learning from the problem].” [8]
- “Often have no idea what to do. Shows to me that they are really not able to read the problem and try to understand the components – I guess that’s “off the scale.” [11]

Several instructors commented on the additional load imposed by the complexity of the language syntax or the environment, particularly if students had not encountered them previously:

- “some get stuck on syntax - if you look at how it’s written, it won’t compile and run, so we find it hard to teach them to pinpoint errors in the code ..
- they are distracted by debugging and ultimately they lose the motivation to look further.” [36]
- “the bottom 10% of students are usually unfamiliar with the programming environment including strict logic, so they have to put in a lot more effort.” [41]
- “they don’t have the [programming] language skills so its about 9, and the rest of it becomes impossible because they get stuck and they can’t go any further.” [44]

- “What we find with the students is if the environment is too complicated then they don’t know the difference between the environment and the language, and if its too simple, then it’s not giving them any help. Like the command line – it’s not a good way of teaching because they have to come to grips with the file structure, and things, so you need an environment that takes away those sort of mechanical elements but still is not full of hundreds of different features that confuse them.” [7]
- “it requires an enormous mental effort because it’s so new, both the language and the environment” [9]

Another broad observation by participants is the general lack of concept generalisation displayed. The high cognitive load experienced by these students is shown in their inability to form schema, and hence to generalise and notice patterns. No transfer is shown by these students:

- “...generalisation … or noticing patterns, is extremely difficult for students. Plenty of times I give them identical problems which to me are identical and then the students see them as completely different problems that are unrelated to previous ones. They just don’t see it. What do they learn from that … ? ” [1]
- “They can do the same problem 4 times in a row and trip over the same bug. Very frustrating actually.” [22]
- “there are students in the bottom portion of the class who will spend a lot more time trying to work out a problem than the average student, but they won’t learn from the experience. And there are students that will.” [38]

The view that students in the bottom 10% of a programming course are simply without the capacity to learn this content, is lamented in a comment by Participant Number 19:

- “the bottom 10% just can’t do it - they flounder” [19]

The final theme offered by instructors concerned the structure of materials or the course expectations. They indicated that in some cases, the bottom 10% of students are expected to fail, and accommodations are not made for these students:

- “most coordinators running these type of courses focus too much on the stronger students: and they should focus on the weaker students.” [41]
- “the bottom 10% of students wouldn’t be expected to pass the unit anyway” [43]
“the assignments are set as a challenge, but not something that is impossible. The good students find them easy, the poor students struggle, and that’s the way it is, we can’t have them too easy or too hard.” [7]

5.2.4 Summary
The comments offered by participants may be summarised as follows:
• many identify multiple profiles of students within the bottom 10% of a programming course;
• some students (Ghosts) are enrolled but never have any intention of attending classes, let alone learning content;
• some students (Idlers) are attending classes, but do not apply themselves to designated tasks (for various reasons) and as such, do not apply sufficient mental effort to enable learning of content;
• some students (Strivers, who were identified by the majority of participants as existing in the bottom 10% of performance within a course) are attending classes, are motivated, are completing (as best they can) designated tasks, are exerting very high to extreme levels of mental effort on everything ... and yet are not demonstrating learning.

We argue that, despite the mental effort measures being based on perceptions by instructors of their student cohort, it is important to note that when probed, many of the instructors made a clear distinction in student failure due to non-engagement with the materials versus those who failed despite their focussed application and effort to the learning resources.

The comments from participants indicate that their perceptions of the Strivers are that they are suffering from excessive mental effort (cognitive overload). Specific themes identified by participants within this profile included:
• Lack of ability in problem solving
• Lack of innate aptitude for programming
• Lack of literacy, comprehension, and analysis skills
• Lack of English due to English being a second language
• Difficulty of the computer language and/or environment being used
• Lack of capacity to generalise concepts
• Instructional materials that do not cater for their needs.

6 Discussion
The results of the current paper are disturbing.
A clear majority (75%) of participants who are academics teaching introductory computer programming courses (units) indicated explicitly the view that there are students in the “bottom 10% of performance” of these courses who are applying themselves to the set learning activities, are completing the assignments (as best they can) and yet they are failing to learn.

Moreover, the reported measures of perceived mental effort by the most frequent (modal) participant response indicates that these very same students who are failing to learn are putting in extreme levels of mental effort. These students cannot be asked to do more.

If these students are to have success in learning computer programming then aspects of the instructional design for introductory programming courses (units) must change.

Many participants indicated that such changes have already occurred as demonstrated by selection of programming language and the increased use of IDEs over the period 2003 to 2010, predominately for pedagogical benefits - that is, to help students’ learning. Yet students still fail to learn, despite their best (mental) efforts.

It is likely that a range of factors contribute to the difficulty of learning computer programming in the current regime. These include the nature of programming itself, the apparent necessity to house activities within a computer language (with associated syntax) and often the inclusion of problem-solving activities.

Cognitive Load Theory warns that the limitations of human cognitive resources means that learning is prone to fail if these resources are not carefully managed in a teaching-learning transaction. The results of the current paper indicate that many students in the lower end of performance within introductory programming courses need to apply large tracts of their cognitive resources to cope with the intrinsic nature of the to-be-learnt content. These students are also required to deploy large, and possibly unnecessary whilst learning basic concepts, levels of cognitive resources to extraneous aspects of instruction, such as the nuances of a programming language and syntax. With such high levels of cognitive resources already deployed it is likely that there is, effectively, nothing left to give to the processes of learning.

Cognitive Load Theory has proven to be an extremely effective utility in engineering better instructional designs for traditionally complex and difficult areas of study such as mathematics, physics and electrical circuitry (see Sweller 1999). Possibilities of improving the instructional design of introductory programming courses may also follow by embracing cognitive analysis of the processes and dynamics associated with learning computer programming.

Perhaps all people associated with teaching introductory programming need to revisit the question “What are the objectives of an introductory programming course?”. The extent to which the answer involves the gaining of core programming concepts, as opposed to skills in any particular language and syntax, may assist in better focussing upon available and effective instructional strategies.

There is also clear theoretical argument that some of the traditional strategies used, such as a focus on problem-solving, may actively block learning. In other technically complex domains such as mathematics (Sweller & Cooper 1985), engineering (Moreno et al. 2006), and psychology (Renkl et al. 2004) the benefits of using worked examples compared to problem solving as a strategy for effective teaching and learning has been demonstrated. It has also been suggested that worked examples and faded worked examples will be beneficial
for teaching introductory programming (Gray et al. 2007) though this remains to be tested.

Greater awareness and understanding of learning through the lens of Cognitive Load Theory, with particular focus on the learning dynamics of novice programmers, may aid in identifying an appropriate response to the design, development and delivery of introductory programming courses.

7 References


Miller, G. (1956): The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. The Psychological Review. 63:81-97


