A Taxonomy of Task Types in Computing

Matt Bower
Macquarie University
Sydney, NSW
Australia
mbower@ics.mq.edu.au

ABSTRACT
Based on the systematic development of a curriculum for our undergraduate computer science units, an analysis of general education and CSE literature and consultation with other computer science educators, a taxonomy of task types in computing is proposed. These task types are related to one another in a hierarchical fashion based on their cognitive interdependencies. The taxonomy can be applied by academics to guide the development of curriculum that meets student process based learning needs rather than just content needs, the latter being the current norm.

Categories and Subject Descriptors
K.3.2 [**Computers in Education**]: Computer and Information Science Education – *computer science education.*

General Terms
Design, Theory.

Keywords
Task Types, Taxonomy, Pedagogy, Computer Science Education

1. INTRODUCTION
Traditionally the Computer Science education community has placed a great deal of emphasis upon defining the type of conceptual knowledge that students need to acquire, and the hierarchical relationship between that knowledge [8, 20, 22, 23]. This represents a pedagogical emphasis on content, which is inevitably perpetuated by teachers who adopt or refer to such curriculum guides. Recent efforts by the Computer Science Learning Taxonomy Working Group [11] commenced shifting emphasis towards programming practices, however at the time of their presentation at ITiCSE07 their work was still in early stages of development and represented one point of view in an area open to many interpretations. Note that this working group has recently published a formalization of their ideas [12].

This paper proposes a Taxonomy of Task Types that identifies the different types of programming processes that students undertake when learning computing and sequences them on the basis of their cognitive interdependencies. In this paper task type is defined as the activity or *process* which students are expected to perform in order to learn the concepts being presented to them. The defining characteristic of the task is what it requires learners to actually do. This can range anywhere from recalling a fact to solving a problem, and a comprehensive hierarchy of such tasks is to follow. Before the taxonomy of task types is proposed, a rationale for such work is provided and literature relevant to the formation of such a taxonomy is presented.

ITiCSE'08, June 30 – July 2, 2008, Madrid, Spain.

2. RATIONALE
The computer science education fraternity has invested a great deal of energy in defining the content that students are expected to learn. There have been major efforts to classify computing in terms of the concepts underpinning the body of knowledge such as the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronic Engineers (IEEE) Computing Curricula joint task force [20]. There have also been related efforts to develop Computer Science ontologies that can be applied for pedagogical purposes [8], as well as efforts to graphically represent the relationships between the various components of knowledge [7]. Significant work has also been performed at the K-12 level [23]. Based on these curriculum resources, classroom educators spend considerable time ensuring that the learning pathway through the content is appropriately sequenced in so far as previous topics cover all required conceptual prerequisites to understand the next concept being addressed. This represents an emphasis on factual and conceptual knowledge (*content*, as described in most computing curricula) which is but one component of computing education. Historically less emphasis has been placed upon the types of tasks (or *processes*) in which students engage [18].

Process is not only crucial because programming is a "skill in practice", but also because process is the means by which factual and conceptual knowledge are formed into well defined schema [2]. Some authors have emphasized the importance of process, not just content. Davies [9] distinguishes between 'programming knowledge' (knowledge of a declarative nature, for example, being able to state how a 'for' loop works) and 'programming strategies' (the way knowledge is used and applied, for example, using a 'for' loop appropriately in a program). Rogalski & Samurcay [1990, cited in 18] make the distinction between cognitive representations or schema (static, 'program as text') in computer programming versus 'plans' (action oriented, 'programming as activity'). Rist [17] has constructed an elaborate model of how programs are generated based on students' underlying knowledge structures, providing a framework for analyzing the various strategies that students use to write computing code.

The taxonomy proposed in this paper provides a resource that can be used to assist the systematic integration of task type into curricula. Such a taxonomy of task types allows dependencies to be identified and appropriations made within a curriculum, based on the corresponding level of task difficulty.

3. RELEVANT WORK

There are several bodies of knowledge and areas of research that have fed into the development of the Taxonomy of Task Types.

Cognitive Science and Educational Researchers generally identify three levels of knowledge [3, 6]:

- *declarative* (or factual) knowledge, such as that relating to the syntax required to write a 'for' loop
- *procedural* knowledge, for instance, how to operate the Integrated Development Environment to debug and compile programs
- *conceptual* knowledge, relating to understanding how the mechanics of a 'for' loop can be used to design a solution to the task at hand.

Conceptual knowledge is founded upon procedural knowledge, which is comprised of related pieces of declarative knowledge. These dependencies provide a basis for ordering the hierarchical levels of the taxonomy, with conceptual knowledge are generally at a higher level than those requiring procedural knowledge, which in turn is higher than those requiring declarative knowledge.

The SOLO taxonomy [4] classifies the level to which learners have been able to relate relevant pieces of information as either being pre-structural, uni-structural, multistructural, relational, or extended abstract. In the SOLO model higher levels of cognition involve greater levels of interrelated knowledge. This is reflected in the taxonomy of computing tasks in so far as higher level task types involve the integration of knowledge and skills acquired at lower levels.

McGill and Volet [14] create a conceptual framework for analysing computing thinking that integrates these three types of thinking from cognitive psychology literature (effectively declarative, procedural and conceptual) with three distinct types of programming knowledge emerging from the educational computing literature (syntactic, semantic, and strategic). These latter dimensions feature within the taxonomy presented in this paper, as tasks requiring more strategic thinking are at a higher level than semantic thinking, which is in turn at a higher level than tasks requiring syntactic thinking.

There have also been efforts to classify programming concepts and thinking in terms of levels of abstractness and generality. For instance, Ahoroni [2] presents a model portraying the level of abstraction from programming languages that students can perform, from Programming Language oriented thinking (low level abstraction, based on the syntax of a language) to Program Oriented Thinking (where reference to a programming language is required, but not necessarily a specific one) to Programming-Free Thinking (high level abstraction, not dependent on aspects of programming languages). Novrat, [15] describes the difference between abstraction and generalization in computing thinking, with abstraction denoting a movement away from levels of detail, as opposed to generalization which is movement up a class hierarchy. Both of these approaches to classifying programming concepts support discrimination between levels of thought, and as such more abstract thinking is represented at more advanced levels of the taxonomy.

One crucial computer science education principle that informs the Taxonomy of Task Types is that of the students' mental model of the computer, or 'notional machine' [18]. The notional machine is "an idealised, conceptual computer whose properties are implied by the constructs in the programming language employed" [10, p. 431]. Robins [18, p. 249] states "the purpose of the notional machine is to provide a foundation for understanding the behavior of running programs". The notional machine is important in discriminating between the types of process in which computing students can engage; while factual recall tasks may not require a well developed notional machine, for practical programming tasks it is essential.

Some general educational taxonomies such as Bloom's [5] and Anderson and Krathwohl's [3] do consider the hierarchical nature of learning processes, such as 'application', 'synthesis' and 'evaluation', and these feed directly into the taxonomy presented in this paper. However because these are built on a generalization of learning that can be applied to any domain they lack a level of focus on processes unique to Computer Science Education such as debugging and system design, and as such a computing specific taxonomy is proposed. It is worth noting that Porter and Calder [16] do briefly make reference to how Bloom's Taxonomy can be applied to the field of computer science education. However their mapping is one-to-one from Bloom's Taxonomy to computer science education whereas the taxonomy presented here expands upon Bloom's Taxonomy and incorporates other literature to devise a taxonomy that is directly based upon tasks in computing.

4. THE TAXONOMY OF TASK TYPES

The taxonomy of computing task types proposed herein includes 10 levels, as follows:

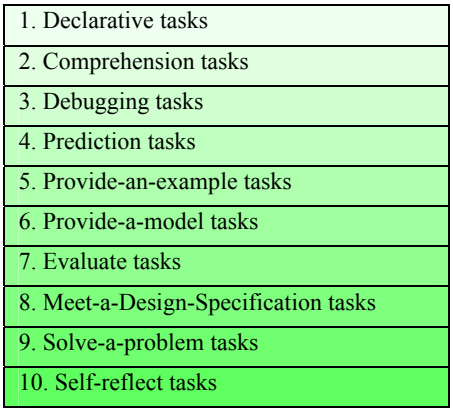

Figure 1 – Taxonomy of Task Types in Computing

The task types represented in Figure 1 have been developed based upon a systematic analysis of existing computer science curricula, consultation with academics from general education and computer science (both intra and extra institutional), and an analysis of educational literature (aforementioned). Specifically, curricula from within our institution were deconstructed, not by conceptual content but by the task they expected students to perform. On this basis the various categories were formed. At the same time general education and computer science education was reviewed to not only provide examples of other tasks, but also to offer insight into other approaches to classifying task

types in computing. Finally, the hierarchy was presented to computer science and education academics for feedback and verification.

It is by no means proposed that this is the only way that task types in computing could be classified, however proposing such a framework provides a catalyst for discussion within the computer science education community and a reference point for comparison to other efforts.

The levels of the taxonomy are now described, along with the rationale for their sequence. Two exemplar tasks are provided up-front for each level so as to offer an illustration of the type as well as an indication of the variety of task types that can fall within each category. While the taxonomy is not absolute in the same way that tasks set at the different levels of Bloom's [5] taxonomy do not automatically ascend in degree of difficulty, it does provide a general hierarchical framework and reference point which computer science educators can use to plan curricula.

5. LEVELS OF THE TAXONOMY

5.1 Declarative Tasks

- *True or false: To include a backslash character '\' in a string you need to 'escape' it by placing another backslash before it.*
- *What is an 'accessor' method?*

Declarative knowledge is static, and usually involves at most one relationship between pieces of information. It is working at the level of recognition and recollection. Declarative tasks are the lowest level of tasks that students can be prescribed, and the knowledge that they embody underpins all tasks at higher levels.

5.2 Comprehension Tasks

- *Explain the difference between the int 127 and the String "127".*
- *Why it is more difficult to test equality for floating-point numbers than integers?*

Typically comprehension involves being presented with an artifact (such as a piece of code) or an item of declarative knowledge and providing an explanation (entire or part). They represent a movement away from straight recall of facts towards tasks requiring an understanding of underlying concepts. Comprehension tasks require students to generate solutions based on an underlying mental model of the concept or situation.

5.3 Debugging Tasks

- *What are the syntactic errors in the piece of code?*

  System.outprntln("Hello);

- *What are the semantic errors within this program?*

```
int i;
int factorial = 1;
for (i = 1; i<=5; i++);
     {factorial = factorial * i;}
System.out.println(i+"! = "+ factorial);
```

Debugging tasks require students to detect errors in programming code, often based upon an anticipation of what the program is trying to achieve. Syntactic debugging tasks rely on well formed declarative knowledge, whereas semantic debugging tasks rely more on well formed comprehensive type understanding. Also, debugging tasks incorporate a significantly greater process aspect than declarative or comprehension tasks, which is another reason that they at a higher level in the taxonomy.

5.4 Prediction Tasks

- *What does this line of code print?*

```
System.out.println("11 + 5" + 20);
```

- *What will be the effect of replacing the `5` with `i+1` in the following code?*

```
public class TwoDtester
{
  public static void main(String[] args)
  {
    int[][] steps = new int [4][];
    for (int i = 0; i<steps.length;i++)
    {
      steps[i] = new int[5];
      for (int j=0; j<steps[i].length; j++)
      {
         steps[i][j] = i+j;
        System.out.print(steps[i][j] + ",");
      }
    }
  }
}
```

Prediction tasks are central to computing because without the ability to predict the effect of the statements comprising a piece of code students cannot write programs. Accurate prediction relies on both accurate comprehension and declarative knowledge. Prediction tasks are generally more cognitively demanding their debugging counterpart, because they rely more heavily on a student's notional machine. As well, Prediction tasks require students to be generative and be able to interpret most all of the code in a program instead of merely identifying particular errors.

5.5 Provide-an-Example Tasks

- *Provide an example of a logical error.*
- *Create an original example of the "dangling else" problem.*

Provide-an-Example tasks are creative tasks that can be either declarative (factual and syntactic) or comprehensive (understanding or semantic) in nature. Note that these generative tasks often demand more intense cognitive engagement than those of previous levels [18]. This is because they either provide smaller cues from which students can retrieve their knowledge (at a declarative level) or require students to synthesise existing pieces of knowledge to create an original representation. This is a higher order thinking capacity that is precursory to the sorts of skills valued by industry.

5.6 Provide-a-Model Tasks

- *Explain what happens behind the scenes in your computer to run a Java program.*
- *Draw a diagram to illustrate what happens in your computer's memory when you:*
  a) *create an object variable (define a new variable name and give its type)*
  b) *initialise that object variable (by creating an object to which it refers).*

This advances beyond Provide-an-Example tasks to not only demonstrate the ability to understand examples that have been

presented, but to also provide an abstract representation of an entire situation or process. This sort of task can be attempted at a fairly low level of cognitive demand if students simply represent models they have found elsewhere. On the other hand, if students are challenged (or challenge themselves) to synthesise their declarative and comprehension knowledge to derive an original model, such tasks can be rich opportunities for relating and restructuring concepts, thus leading to deeper understanding. This sort of task requires declarative and comprehension knowledge, and often involves linking the two. Providing an explanatory model can often improve students' debugging and predictive knowledge by developing their notional machine.

5.7 Evaluate Tasks

- *Evaluate the following method as an approach to providing the value of the username field of the User class:*

```
public String getUserName ()
{
  System.out.println("The username is: "
                      + username);
 return username;
}
```

- *What are the advantages and disadvantages of having types in a programming language?*

Evaluation tasks, which have previously often been associated with the highest order of thinking in taxonomies such as Bloom's [5], can actually be pitched at a great variety of difficulty levels and can achieve many different types of thinking. For instance, the task "provide a list of the advantages of applets over applications" will often result in the reproduction of a text book response or information found on the Internet. However this task can also be approached at an expert level when attempting to decide which approach to adopt to roll out a tool to customers. Evaluation can occur upon the final solutions presented for Provide-an-Example and Provide-a-Model tasks. The subjective or "soft knowledge" nature of evaluative tasks is useful in so far as such activities can usually be attempted by students of lower abilities, but are also open-ended enough to stimulate more capable students. These sorts of tasks are perfectly suited to collaborative approaches because all students can contribute and the weaker students can benefit from being exposed to the thought processes of the more able pupils.

5.8 Meet-a-Design-Specification Tasks

- *Write a program that uses a 'for' loop to print out all the even numbers between 100 and 2 in reverse order, i.e., 100, 98, 96, etc.*
- *Design a system to meet the following specification:*
  The system contains Lecturers, UnderGradStudents and PostGradStudents.
  a) *Every Person in the system has a name.*
  b) *The system also holds:*
     i) *whether a Lecturer is at a senior level 'S' or a normal level 'N'.*
     ii) *the student number of each Student*
     iii) *the fee structure of each Student (assume "Full Fee" for PostGradStudent, "HECS" for UnderGradStudent)*
     iv) *a String outlining the previous qualification of a PostGradStudent*
     v) *whether an UnderGradStudent wants to be part of the mentorship program ('Y' or 'N').*

Design tasks require students to combine their knowledge to present an original and creative solution. Design tasks can be pitched at an implementation level (e.g. write the code), or a conceptual level (e.g. provide a UML class diagram). Implementation level design requires the underlying cognitive skills developed through declarative, comprehension, debugging, prediction, and providing example tasks. Conceptual level design requires only declarative and comprehension knowledge. Good conceptual design requires pragmatic understanding developed in debugging, prediction, example creation and evaluation tasks. That is to say, students can create models of systems without understanding how to implement them (ie, provide diagrams and method names), but without an appreciation of what is required to implement the system their capacity to construct expert designs is restricted. Whereas evaluation of Provide-an-Example and Provide-a-Model tasks is more summative (subsumes these tasks), evaluation should be an ongoing process that occurs during Meet-a-Design-Specification tasks (is contained within such tasks). For this reason Meet-a-Design-Specification tasks have been placed at a higher level on the taxonomy than Evaluation tasks.

5.9 Solve-a-Problem Tasks

- *What is the smallest number that has fifty different factors?*
- *Diana wants to check whether her students' test scores seem consistent with their assignment marks. Create a system that aids her attempts to do so.*

Solve-a-Problem tasks could be considered another way of framing design tasks, however because they require the student to respond to more ill-structured task requirements they have been classified as a separate category. Some students may be able to design systems to meet specifications, but less capable at solving problems where the approach is not well defined. Solve-a-Problem tasks promote the development of context scoping, critical thinking, and cognitive flexibility, which are in the realm of more expert behaviors [1, 21].

5.10 Self-Reflect tasks

- *Reflect on the way that you have attempted to complete this task/attempted to learn this body of knowledge.*
- *Reflect on the way that you have engaged with others in this process.*

Self-Reflect tasks require students to evaluate the ways in which they learn (as opposed to evaluating subject matter content). Reflection can be based on one's own engagement with the content or process (i.e., metacognitive). This aids the development of control skills – the capacity to self-monitor and evaluate decisions made during the problem solving process – which is a key determinant of problem solving performance [13, 19]. As well, reflection can also be on one's engagement with others (which may or may not relate to the collaborative design or content of the task) in an attempt to improve one's ability to learn from and with others. While reflection does not necessarily incorporate skills required in all lower levels of tasks, it can be applied to all previous levels. For this reason it has been included as the highest level in this taxonomy.

6. CLOSING WORDS

The hierarchy presented does not imply that higher level tasks should be left to the end of an undergraduate computing course and that tasks at lower levels should only occur at the beginning. It is important that students at early stages of learning computing are encouraged to perform tasks that foster higher order thinking, albeit on a smaller scale and focusing on less complex material than at later stages of their computing education.

Nor is it proposed that that all tasks will neatly fall within one level of the Taxonomy; often task types can be prescribed in combination (for example "Describe the dangling else problem and compose an original example that illustrates it").

What is being proposed is that reflecting on the levels of the Taxonomy during curriculum planning may support a worthwhile shift in focus. Programming process has often been relatively ignored when constructing curricula in favor of content. Yet most Computer Science Educators agree that computer programming is a practice and not just a body of knowledge. To this extent it would seem natural to define and create curriculum based at least in part based upon the tasks that student have to perform, their dependencies and relationships.

The taxonomy of computing tasks proposed in this paper draws to the educator's consciousness the range of tasks in which students should engage in order to achieve mastery in the area. The way in which the levels are hierarchically related highlights the need for incremental and considered deployment of task types in a manner that accounts for the prerequisite skills of the learners. In that way, along with content, task type can be used as a means of adjusting the cognitive demands at different stages of the curriculum, providing more effective learning sequences for students.

7. REFERENCES

[1] Agnew, N. M., Ford, K. M., and Hayes, P. J., *Expertise in Context: Personally Constructed, Socially Selected and Reality Relevant?*, in *Expertise in Context*, P.J. Feltovich, K.M. Ford, and R.R. Hoffman, Editors. 1997, AAAI Press / The MIT Press: Melno Park, 219-244.

[2] Aharoni, D., Cogito, Ergo sum! cognitive processes of students dealing with data structures. *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, (2000), 26-30.

[3] Anderson, L., and Krathwohl, D., *A Taxonomy for Learning, Teaching and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Longman, New York, 2001.

[4] Biggs, J., and Collis, K., *Evaluating the Quality of Learning: the SOLO taxonomy*. Academic Press, London, 1982.

[5] Bloom, B. S., *Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain*. David McKay Co Inc, New York, 1956.

[6] Byrnes, J. P., *Minds, Brains and Learning*. The Guilford Press, New York, 2001.

[7] Cassel, L. N., Hacquebard, A., McGettrick, A., Davies, G., LeBlanc, R., Riedesel, C., Varol, Y. L., Finley, G. T., Mann, S., and Sloan, R. H., ITiCSE 2005 working group reports: A synthesis of computing concepts. *ACM SIGCSE Bulletin*, *37*, 4 (2005), 162-172.

[8] Davies, G., Cassel, L. N., and Topi, H. Using a computing ontology for educational purposes In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education ITICSE '06* 2006, 334.

[9] Davies, S. P., Models and Theories of Programming Strategy. *International Journal of Man-Machine Studies*, *39*, 2 (1993), 237-267.

[10] du Boulay, B., O'Shea, T., and Monk, J., *The black box inside the glass box: presenting computing concepts to novices*, in *Studying the Novice Programmer*, E. Soloway and J.C. Spoher, Editors. 1989, Lawrence Erlbaum: Hillsdale, NJ, 431-446.

[11] Fuller, U., and Johnson, C., *Working Group Report: Developing a Computer Science-Specific Learning Taxonomy* in *12th annual SIGCSE conference on Innovation and technology in computer science education ITICSE '07*. 2007: Dundee, Scotland.

[12] Fuller, U., Johnson, C. G., Ahoniemi, T., Cukierman, D., Hernán-Losada, I., Jackova, J., Lahtinen, E., Lewis, T. L., Thompson, D. M., Riedesel, C., and Thompson, E., Developing a computer science-specific learning taxonomy. *ACM SIGCSE Bulletin*, *39*, 4 (2007), 152-170.

[13] Ginat, D., Metacognitive awareness utilized for learning control elements in algorithmic problem solving. *SIGCSE Bull.*, *33*, 3 (2001), 81-84.

[14] McGill, T. J., and Volet, S. E., A Conceptual Framework for Analyzing Students' Knowledge of Programming. *Journal of Research on Computing in Education*, *29*, 3 (1997), 276-297.

[15] Novrat, P., Hierarchies of programming concepts: abstraction, generality, and beyond. *SIGCSE Bull.*, *26*, 3 (1994), 17-21.

[16] Porter, R., and Calder, P. Patterns in Learning to Program - An Experiment? In *Proc. Sixth Australasian Computing Education Conference (ACE2004)* 2004, 193-199.

[17] Rist, R. S., Program Structure and Design. *Cognitive Science*, *19*, (1995), 507-562.

[18] Robins, A., Roundtree, J., and Roundtree, N., Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, *13*, 2 (2003), 137-172.

[19] Schoenfeld, A., *Mathematical Problem Solving*. Academic Press, New York, 1985.

[20] Shackelford, R., McGettrick, A., Sloan, R., Topi, H., Davies, G., Kamali, R., Cross, J., Impagliazzo, J., LeBlanc, R., and Lunt, B. Computing Curricula 2005: The Overview Report In *Proceedings of the 37th SIGCSE technical symposium on Computer science education SIGCSE '06*. (Houston, Texas, USA). 2006, 456-457.

[21] Spiro, R. J., Coulson, R. L., Feltovich, P. J., and Anderson, D. Cognitive flexibility theory: Advanced knowledge acquisition in ill-structured domains. In *Proceedings of the 10th Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum 1988.

[22] The Joint Task Force on Computing Curricula, Computing Curricula 2001. *Journal on Educational Resources in Computing (JERIC)*, *1*, 3 (2001), 1-236.

[23] Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., and Verno, A. (2003) A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Education Task Force Computer Science Curriculum Committee. Last accessed October 2006 [Available at: http://www.csta.acm.org/Curriculum/sub/ACMK12CSModel.html]