# Bloom's Taxonomy for CS Assessment

**Errol Thompson**
2 Haven Grove, Naenae,
Lower Hutt
New Zealand
E.L.Thompson@massey.ac.nz

**Andrew Luxton-Reilly**
Computer Science
University of Auckland
Private Bag 92019, Auckland,
New Zealand
andrew@cs.auckland.ac.nz

**Jacqueline L. Whalley**
Computer and Information Sciences
Auckland University of Technology
Private Bag 92006, Auckland 1020,
New Zealand
jacqueline.whalley@aut.ac.nz

**Minjie Hu**
Tairawhiti Polytechnic
PO Box 640,
Gisborne,
New Zealand
min@tairawhiti.ac.nz

**Phil Robbins**
Computer and Information Sciences
Auckland University of Technology
Private Bag 92006, Auckland 1020,
New Zealand
phil.robbins@aut.ac.nz

## Abstract

Bloom's Taxonomy is difficult to apply consistently to assessment tasks in introductory programming courses. The Bloom taxonomy is a valuable tool that could enable analysis and discussion of programming assessment if it could be interpreted consistently. We discuss each of the Bloom classification categories and provide a consistent interpretation with concrete exemplars that will allow computer science educators to utilise Bloom's Taxonomy for programming assessment. Using Bloom's Taxonomy to help design examinations could greatly improve the quality of assessment in introductory programming courses.

*Keywords:* Bloom's taxonomy, multi-institutional, novice programmers

## 1 Introduction

Bloom's taxonomy was first described as a hierarchical model for the cognitive domain in 1956 (Bloom et al. 1956). The model was revisited in 2001 by Anderson and a team of cognitive psychologists. As a result, a number of significant changes were made to the terminology and structure of the taxonomy (Anderson et al. 2001). These two versions of the taxonomy of educational objectives are often referred to as Bloom's taxonomy (Bloom et al. 1956) and the revised Bloom's taxonomy (Anderson et al. 2001).

Bloom's taxonomy has been applied to the education domain of computer science for course design and evaluation (Scott 2003), structuring assessments (Lister et al. 2003) and comparing the cognitive difficulty level of computer science courses (Oliver et al. 2004).



Some attempts have been made to relate Bloom to specific computer programming tasks. Abran et al. (2004) used Bloom's taxonomy to classify typical programming and software engineering tasks. Schneider and Gladkikh (2006) used the revised Bloom's taxonomy for planning diagnostic assessments for programming, systems analysis and systems design.

Johnson and Fuller (2006) and a team of academic colleagues examined the question 'Is Bloom's Taxonomy Appropriate for Computer Science?' They attempted to determine which cognitive process, in the revised Bloom taxonomy, was targeted by different assessment tasks. They noted significant disagreement between academics who had been involved in the teaching of the course and those who had not. They suggested that the disagreement was due to two factors. Firstly, intimate knowledge of the way a course is taught is required in order to accurately determine the cognitive process targeted by some assessments; and secondly, there was no general agreement about how to apply the Bloom taxonomy to tasks in computer science.

Previously we reported on work that aimed to add new problems (Whalley et al. 2006) to the Leeds programming comprehension research toolkit (Lister et al. 2004). Initially we focused on developing a research instrument that would allow us to duplicate and extend the Leeds toolkit. Because we were aiming to measure student comprehension of code, we set out to design a set of questions that lay within the Understand sub-categories of the cognitive dimension of the revised Bloom's taxonomy.

Although the revised Bloom's taxonomy was useful in formulating ideas for writing program comprehension questions, we found that it was often difficult to formally categorise a completed question within the cognitive dimension. Moreover we found it easier to categorise questions at the category level than at the subcategory level within the cognitive dimension.

We believe that it is important to develop a common understanding of how the revised Bloom's taxonomy is interpreted in the domain of computer science. In this

paper we provide an interpretation of the taxonomy as it applies to introductory programming exams. We have chosen to base our categories on the two-dimensional matrix of the revised taxonomy that relates the cognitive process dimension to a knowledge dimension. In this paper we will limit the discussion to the top level of the cognitive process dimension. The analysis of the knowledge dimension will be discussed in a future paper.

## 2 Methodology

For this study, exam scripts from first-year programming courses were supplied by 6 institutions from Australasia and the USA. The questions in the exams varied in nature and included true or false, multiple-choice, short and long answer questions. Each exam script was independently analysed by the 5 authors, and its questions classified according to the revised Bloom's taxonomy. The exams were all written final exams, and each individual question was classified in both the cognitive and knowledge dimensions.

Following this first classification phase the analysers met to discuss the application of the revised Bloom's taxonomy. Differences in the way that each academic applied the taxonomy were noted and discussed in detail in order to determine the cause of the discrepancy and to come to a common understanding.

Initially we discovered significant differences between the categories that we had assigned to many questions. This was primarily due to difficulty mapping the cognitive tasks described by the taxonomy's authors into the programming domain, for which there are no examples.

In some cases, differences in categorisation were due to an academic being involved with a course and therefore able to provide the teaching context for the assessment task in question. Once the teaching context was elucidated, we were able to agree on an appropriate cognitive category for the assessment task in question.

The following question provides an example of this process.

*Given the following class:*

```
public class Circle
{
    private int diameter;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    public Circle()
    {
        diameter = 30;
        xPosition = 20;
        yPosition = 60;
        color = "blue";
        isVisible = false;
    }

    //code removed for bevity
}
```

*Write a constructor that would allow the location, colour, and diameter of the circle to be set. Show how this constructor would be used to create a circle at x = 200 and y = 400, with colour blue, and diameter = 90.*

The authors who were not involved in the teaching of the course categorised this question either as **Understand** (on the basis that this question required students to provide an example of a familiar concept), or **Create** (on the basis that it asked students to combine code in a way that they had not seen before).

The person who had taught the course classified the question as **Apply**. The course material explicitly taught a process for writing constructors that accepted parameters. The lecturer of the course felt that this question asked students to apply a known process to a familiar situation (i.e. the students had been taught a process for handling this sort of question and had seen similar examples, but had not seen this particular code).

Once the teaching context had been explained, the authors agreed unanimously that **Apply** was the appropriate classification in this case. We concluded that in order to effectively analyse a question the person undertaking the analysis should have an in-depth knowledge of the course as a whole. This belief is supported by Bloom et al. (1956), Anderson et al. (2001), and Johnson and Fuller (2006).

Using the analysis as a talking and reference point, the authors developed an agreed understanding of the Bloom categories and subcategories and developed new descriptors. Using these new descriptors each author re-analysed their own exam papers, for which they had an intimate knowledge of the course content.

## 3 Cognitive Categories

Anderson et al. (2001) provide vignettes of how the knowledge and cognitive categories apply in a number of different subject area domains. Computer science and programming are not among the subject areas covered. Here we endeavour to describe the categories using examples specific to programming.

One of the difficulties with using the cognitive hierarchy in a programming context is clarifying what it means to apply a process and/or to create a process. For the purposes of this paper, the following distinction is made.

**Process:** This is the procedure that a person might learn or create in order to be able to write a code segment. Examples of processes are code tracing, desk checking, translation from design to code, and implementing a known algorithm. In terms of the knowledge dimensions of the taxonomy this is process knowledge.

**Algorithm:** This is used in the computer science sense as a portion of program code or a code pattern designed to achieve a particular task within a program. From an object-oriented perspective, a design pattern would be the equivalent of an algorithm. This is also regarded as process knowledge within the taxonomy (Anderson et al. 2001).

## 3.1 Remember

**Remember** is defined as 'retrieving relevant knowledge from long-term memory' (Anderson et al. 2001). In the revised taxonomy, this category includes recognising and recalling. We interpret this in programming assessment terms to mean:

1. identifying a particular construct in a piece of code;
2. recognising the implementation of a subject area concept;
3. recognising the appropriate description for a subject area concept or term;
4. recalling any material explicitly covered in the teaching programme. This might be factual knowledge, the recall of a conceptual definition, the recall of a process, the recall of an algorithm, the recall of a design pattern, or the recall of a particular algorithm or design pattern implemented as a solution to a specific problem in exactly the same context as a classroom based exercise.

### Examples

a) **List** the arithmetic operators in increasing order of precedence.

b) **Define** the purpose of a constructor.

c) **Describe** the state pattern.

### Discussion

In these instances students are asked to perform tasks requiring knowledge that they could have rote-learned. The use of verbs such as **list** and **describe** are regarded as synonyms for recall. In the second example above, the task would belong to the **Remember** category if the course materials included a definition of the purpose of a constructor (for example, on an overhead slide).

Determining if a task belongs to this category often requires detailed knowledge of the course materials, since the most significant factor for this category is whether the student has seen the solution to the task before. If the task can be completed simply by remembering something, the assessment task belongs to this category; otherwise it must belong to one of the following 5 categories.

## 3.2 Understand

**Understand** is defined as 'constructing meaning from instructional messages, including oral, written, and graphical communications'. In the revised taxonomy, this category includes Interpreting, Exemplifying, Classifying, Summarising, Inferring, Comparing, and Explaining. We interpret this in programming assessment terms to mean:

1. translating an algorithm from one form of representation to another form;
2. explaining a concept or an algorithm or design pattern;
3. presenting an example of concept or an algorithm or design pattern.

### Example one

Look at this section of code and explain in plain English what it does.

```
public static int mystery(int[] x, int a, int b)
{
  int z = 0;
  for (int i = a; i <=b; i++)
  {
    z = z + x[i];
  }
  return (z / (b-a+1));
}
```

### Discussion

The students are provided with a segment of code and asked to explain what the code does. **Explain** is one of the subcategories of the **Understand** category.

### Example two

The students have been provided with the source code for a class. They are asked to:

a) Identify the constructor(s) defined in this class by writing constructor signatures in the answer book.

b) Write a statement that would instantiate (create) an object using the constructor(s) that they have identified. Write any additional Java code that would help clarify the data type of any variables involved.

### Discussion

This example targets two distinct cognitive process categories. Before students are able to identify a given programming construct (such as a constructor), they must **recall** the syntax rules for that construct and use those rules to **recognise** that construct in the provided code. This portion of the question belongs in the **Remember** category.

Having identified the constructor, the students are then asked to "write" a statement that instantiates an object. Write is not directly mapped to any of the cognitive process categories, so we need to look at what is involved in this activity. In this case, the students must infer what an appropriate calling sequence is, based on the signature of the identified constructor. Inferring is a subcategory of the **Understand** cognitive process category.

## 3.3 Apply

**Apply** is defined as 'carrying out or using a procedure in a given situation'. In the revised taxonomy, this category includes Executing and Implementing. We interpret this in programming terms to mean:

1. that the process and algorithm or design pattern is known to the learner and both are applied to a problem that is familiar, but that has not been solved previously in the same context or with the same data or with the same tools; or
2. that the process and algorithm or design pattern is known to the learner, and both are applied to an unfamiliar problem

### Example one

**Evaluate** the expression: 2 + 4 / 7 * 5 % 3 == 7

### Discussion

This example requires a student to follow a known process and to apply the rules of precedence in order to evaluate the expression shown. If the expression was extremely simple, such as "1 + 2", then we would expect the student to evaluate the expression using recall, so the **Remember** category would be most appropriate. In this less simple case, the complexity of the expression requires students to follow an algorithm in order to compute the results. The process requires students to understand the rules dictating the order of precedence and evaluate the expression by performing the operations in the correct order. The critical part of the question that results in the **Apply** categorisation is that students are applying a process in order to solve the problem (in this case, applying a known process to a familiar problem, although with unfamiliar data).

Although the word "Evaluate" is used in this question, the meaning is not the same as the meaning of the cognitive process category **Evaluate** which is "making judgements based on criteria and standards" (Anderson et al. 2001). This isn't what the students are being asked to do in this example. "Evaluate" in this context means to apply the process for expression evaluation to determine the end result of using the given expression. This example is therefore in the **Apply** cognitive process category.

### Example two

The students have been given the code for a Circle class. The code is similar to an example used in the textbook but modified to reduce the amount of code and change some features.

As well as the *Circle* class, the project includes *Square* and *Triangle* classes. Each class has the same code structure. Students are asked to:

a) Create a *Shape* class as a superclass of these three classes that includes all the common methods.

b) Rewrite the *Circle* class to inherit from the new *Shape* class.

### Discussion

This example belongs to the **Apply** category because the students have been introduced to the process of refactoring. They are expected to apply the refactoring process to develop (create) a shape class and then a revised (rewrite) Circle class. The use of the verbs create and rewrite in this context does not imply being creative in the sense of the **Create** category: students are not being expected to develop a new process or a new algorithm.

## 3.4 Analyse

**Analyse** is defined as 'breaking material into its constituent parts and determining how the parts relate to one another and to an overall structure or purpose'. In the revised taxonomy, this category includes Differentiating, Organising, and Attributing. We interpret this in programming assessment terms to mean:

1. breaking a programming task into its component parts (classes, components, etc.);
2. organising component parts to achieve an overall objective;
3. identifying critical components of a development;
4. identifying unimportant components or requirements.

### Example

Given the code for a Circle class, the students are asked:

a) What is the method *Circle* in this class?

b) How does it differ from other methods in the class?

### Discussion

In the example above, students were expected to provide answers such as a) "It's a constructor", and b) "It is invoked when a new objected is created". This is the reverse of the question used as example two for the **Understand** category. Given the name of the method, the students have to identify what type of method it is, and then identify the difference between it and other methods. The first part of the question (what is) involves recalling that a method with the same name as the class is a constructor, and concluding that the named method is therefore a constructor. In the second half of the question (how does) the students are being asked to differentiate between a constructor and other methods of the class. Differentiating is one of the subcategories of the **Analyse** cognitive process category.

## 3.5 Evaluate

Evaluate is defined as 'making judgements based on criteria and standards'. In the revised taxonomy, this category includes Checking and Critiquing. We interpret this in programming assessment terms to mean:

1. determining whether a piece of code satisfies the requirements through defining an appropriate testing strategy;
2. critiquing the quality of a piece of code based on coding standards or performance criteria.

### Example

The students have been given a class that has the following declaration.

```
private double numbers[] = new double[10];
private int used = 0;
```

In that class, there is an existing method that calculates the minimum using the following for loop.

```
for (int i = 0; i < used; i++) {
  min = Math.min(min, numbers[i]);
```

The question reads:

It has been proposed that a better solution for the min method would be

```
public double min() {
  double min = numbers[0];
  for (double number : numbers) {
    min = Math.min(min, number);
  }
  return min;
}
```

Discuss the differences between these solutions using the current collection type of the *numbers* variable and discuss which method is more appropriate for the current collection type.

### Discussion

Discussing the differences involves comparing the two loop constructs and contrasting their usage. This belongs in the **Understand** category. The students are asked to go further and to discuss which method is more appropriate. This involves evaluating the use of two different loop constructs that are used for the same purpose. The second option fails because all cells in the array will be used in finding the minimum even if some of the cells of the array have not been given values. The students must use this knowledge to evaluate the preferred loop construct for the given collection type. This question is therefore in the **Evaluate** cognitive process category.

## 3.6 Create

Create is defined as 'putting elements together to form a coherent or functional whole; reorganising elements into a new pattern or structure'. In the revised taxonomy, this category includes Generating, Planning, and Producing. We interpret this in programming assessment terms to mean:

1. coming up with a new alternative algorithm or hypothesising that a new combination of algorithms will solve a problem;
2. devising an alternative process or strategy for solving a problem; or complex programming tasks, this might include dividing the task into smaller chunks to which they can apply known algorithms and processes;
3. constructing a code segment or program either from an invented algorithm or through the application of known algorithms in a combination that is new to the students..

### Example

Write a method `get24HourTime()` which accepts three parameters and returns a `String`. The three parameters are an `int` representing the hour value, an `int` representing the minute value and a `String` which is either "`am`" or "`pm`". The method returns a `String` representing the time as a 24-hour time value. For example, 2:35pm is "14:35" in 24-hour time.

**Note:** 12:0pm is "12:0" in 24-hour time and 12.0am is "0:0" in 24-hour time.

For example, executing the `Q4` program with the completed `get24HourTime()` method produces the following output:

```
> java Q4App
20:23
12:0
0:0
7:15
```

### Discussion

The difficulty with questions of this type is to determine whether they are **Apply** or **Create**. The size of the problem does influence the difficulty of the problem, but it doesn't determine whether it is **Apply** or **Create**. The **Create** category should require creative thinking and the formation of a "coherent or functional whole" (Anderson et al. 2001). If the students are familiar with the algorithm and process then the cognitive load is lower and therefore the question should be categorised as **Apply**. To answer this type of question, the students should be familiar with the process for designing an algorithm.

The cognitive category of **Create** applies where the student has no familiarity with completed functional whole. While they haven't seen the algorithm before, they might have seen background material or bits and pieces, but not the completed whole.

The cognitive category of **Apply** requires knowledge of an algorithm and/or process and its application to a given situation. In programming terms this is where students have seen the same or a very similar algorithm working with different data or presented in a different implementation language.

The cognitive category of **Remember** could apply to this type of question if the students had already seen the exact problem solution in the same language, algorithm and process. That is, they have seen the exact same thing in the same context.

In a large program there may be parts that are **Apply** (i.e. applying a design pattern) but the whole could be **Create** since there may be a need to use novel strategies and coding as a link between the component parts.

With this example, the authors were informed by the teacher that the students had seen other reformatting exercises but not this specific exercise. They were familiar with the concepts of reformatting but not of the specific algorithm or process. The question was therefore categorised as **Create**.

## 4 Analysis and Discussion

Using the revised Bloom's taxonomy forced us to review the exam questions in terms of how the paper/subject was taught. Simply reading the questions did not always give a clear indication of the cognitive skill involved in addressing the question. In part, this is caused by the use of verbs like *write*, *create*, and *evaluate* in writing programming exam questions. Once staff involved in the teaching of a course were consulted, we found considerable agreement in the categorisation of questions according to the Bloom taxonomy. We consider this to be a positive outcome which suggests that the revised Bloom's taxonomy can be effectively used to discuss examination questions in the programming domain.

In determining the cognitive skill level required for a question, the level of difficulty of the question is not a factor. For example, some questions requiring students to recall something covered in class would be extremely easy (such as "What language do we use to program in this course?"), while others would be extremely difficult (such as "What is the 3rd word that appears on slide 3 of the second lecture?").

It should also be recognised that the actual cognitive process that is applied to a specific task will depend on the individual solving that task. A given task might require nothing more than recall (the lowest level of cognitive process) for one individual, but may require another individual to generate a new solution to a situation that they find novel (using the highest level of cognitive process). The context is critical for assessing the level of process that we think *most* students will require in order to answer a given question.

During the analysis of the examinations, we found examples of questions that could be reworded in such a way that the cognitive level is altered. For example, a question that was considered to be operating at the **Remember** cognitive level could be reworded so that it required an answer that involved understanding or analysis. We felt that a shared understanding of the interpretation of the revised Bloom's taxonomy to the programming domain would prove valuable to teaching staff developing examination questions, particularly in courses that involve multiple staff members.

## 5 FUTURE WORK

The Bloom taxonomy focuses on knowledge categories and the cognitive skills utilised. This gives one approach to analysing the difficulty of question sets. In this analysis we focused on the categories and not the subcategories. We are currently refining these definitions to cover the subcategories. The revised Bloom taxonomy provides a two-dimensional matrix in which an assessment task is mapped to a category in both a cognitive process dimension and a knowledge dimension. We intend to look more closely at the applicability of the knowledge dimension to the programming domain and identify how a common understanding of the knowledge dimension can contribute to the development and analysis of assessment tasks in computer science education.

An alternative approach is the structural approach proposed by Biggs and Collis (1982) for the analysis of student responses to questions. A question that has been classified as **Understand** (e.g. explain in plain English a segment of code) can be aimed at a number of the categories of the SOLO taxonomy. We are presently undertaking further investigation to see how this taxonomy might be applied to the writing of exam questions.

## 6 Conclusion

We have provided an interpretation of the revised Bloom's taxonomy for computer science. We feel that it is important for the discipline of computer science to look carefully at the cognitive processes that programming requires. The use of Bloom's taxonomy provides some insight into these processes. The provision of a common understanding of the taxonomy for programming enables discussion around assessment and cognitive processes. We hope that this paper generates discussion and more critical analysis of our assessment tasks in programming.

## 7 Acknowledgments

Thanks to Raymond Lister (UTS, Sydney, Australia) and Beth Simon (UBC, Vancouver Canada) for generously providing their exam scripts for this analysis.

## References

Abran, A., Moore, J., Bourque, P., DuPuis, R. and Tripp, L. (2004) Guide to the Software Engineering Body of Knowledge - 2004 Version SWEBOK®, Los Alamitos, CA , IEEE-CS - Professional Practices Committee.

Anderson, L.W., Krathwohl, D.R., Airasian, P.W., Cruikshank, K.A., Mayer, R.E., Pintrich, P.R., Raths, J. and Wittrock, M.C. (eds.) (2001). *A taxonomy for learning and teaching and assessing: A revision of Bloom's taxonomy of educational objectives*. Addison Wesley Longman.

Biggs, J.B. and Collis, K.F. (1982) *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. New York, Academic Press.

Bloom, B.S., Engelhart, M.D., Furst, E.J., Hill, W.H. and Krathwohl, D.R. (1956) *Taxonomy of educational objectives Handbook 1: cognitive domain*. London, Longman Group Ltd.

Johnson, C.G. and Fuller, U. (2006) Is Bloom's taxonomy appropriate for computer science. Berglund, A. ed. *6th Baltic Sea Conference on Computing Education Research (Koli Calling 2006)*, Koli National Park, Finland, 115-118.

Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J.E., Sanders, K., Seppälä, O., Simon, B. and Thomas, L. (2004) A multi-national study of reading and tracing skills in novice programmers. *Inroads - The SIGCSE Bulletin*, **36** (4). 119-150.

Lister, R. and Leaney, J. Introductory programming, criterion-referencing (2003) *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, 143-147, ACM Press.

Oliver, D., Dobele, T., Greber, M. and Roberts, T. (2004), This course has a Bloom Rating of 3.9. in *Proceedings of the sixth conference on Australasian computing education - Volume 30*, Dunedin, New Zealand, 227-231, Australian Computer Society Inc.

Scott, T. (2003) Bloom's taxonomy applied to testing in computer science classes. *Journal of Computing in Small Colleges*, **19** (1). 267-274.

Shneider, E. and Gladkikh, O. (2006) Designing questioning strategies for information technology courses. Mann, S. and Bridgeman, N. eds. *The 19th Annual Conference of the National Advisory Committee on Computing Qualifications: Preparing for the Future — Capitalising on IT*, Wellington, 243-248, National Advisory Committee on Computing Qualifications.

Whalley, J., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, A. and Prasard, C. (2006), An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies. in *Eighth Australasian Computing Education Conference (ACE2006)*, Hobart, Tasmania, Australia, CRIPT, **52**, 243-252., Australian Computer Society Inc.