A Comparison of Constructivist VS Behaviourist Assignment Sets for CS102

J. R. Parker
Digital Media Laboratory
Department of Computer Science
University of Calgary
2500 University Dr. N.W.
Calgary, Alberta, Canada T2N 1N4
1 403 220 6784
parker@cpsc.ucalgary.ca

Katrin Becker
Computer Science Education Group
Department of Computer Science
University of Calgary
2500 University Dr. N.W.
Calgary, Alberta, Canada T2N 1N4
1 403 220 6784
becker@cpsc.ucalgary.ca

Abstract

Two approaches to teaching Computer Science are compared, using two sets of assignments given to distinct CS102 lecture sections during the same semester. The complexity and effort represented by the solutions is compared using software engineering metrics, giving a measure of the effectiveness of the two assignment sets.

Keywords

CS102, computer science instruction, constructivist learning

1 Introduction

Constructivism is a learning model that is becoming very popular in post secondary institutions in North America. It maintains that learning must be active, and is not just about the discovery of facts [13,3,4]. Teachers must guide the learners in the construction of mental models. Constructivism is about helping the learner construct a viable model, and the guidance is based on the individual learner's currently existing cognitive structures. One practical way of introducing constructivist methods into a program is to introduce an experiential learning component.

The University of Calgary has, for the past three years, been encouraging the incorporation of an experiential learning component in all of the undergraduate programs. In the Winter session of 2002 an unfortunate situation developed that allowed the Department of Computer Science to test the efficacy of this strategy, at least in part. Because of an administrative problem, different sections of the same course - Computer Science 233, equivalent to the ACM CS102 course - were allowed to proceed individually, with no coordination of assignments, lectures, or exams. This was unfortunate, but it did allow us to conduct a simple comparison of the methods used in the two courses; specifically, the assignments given to the students will be assessed and compared.

Students submit their assignment solutions using an on-line submit program, and this allowed us to collect all of the submissions as they were made. The intention was to use standard software engineering complexity metrics to determine how complex the solutions were, and to compare this between the two distinct lecture sections. Solutions that reflect a high degree of sophistication, complexity, and effort represent a better educational experience, at least according to most learning models. Learning to program is learning to construct mechanisms and explanations[17]; the hard problem for novice programmers is not in the constructs of a language, but in putting the pieces together. Students need to be taught about typical solutions to problems and strategies for using them.

In the remainder of this paper we discuss two educational models represented by the two sets of assignments: the *behaviourist* model and the *constructivist* model. First, previous work in this area is summarized. The assignment sets we used are then described, and the measures applied to the solutions are explained in detail. Finally the results of the measurements are given, and the conclusions supported by the results are presented.

2 Related Work

There appears to be very little previous work [5,6] that involves a complexity analysis of assignments in the way that is being done here. There is a body of work on the use of constructivism, its nature, and its advantages and disadvantages (E.G. [1,16]). There are other learning models, and a body of literature concerning each one. We are, of course, primarily interested here in learning models in Computer Science education and their relationship to programming assignments.

2.1 Making Quality Count in Undergraduate Education

Romer[15] establishes that expectations should be high, but attainable, and should be clearly communicated at the start. It is important that attainability be extended to the vast majority of the students, and not just the top few percent.

The first year is critical for student success, and we should respect the diverse learning styles of the students from the outset. He maintains that students must be able to synthesize experiences from different contexts in a single problem. There must also be opportunities for collaborative learning, and the students must have out of class contact with faculty[1,16].

2.2 Simulations, Games, and Experience based Learning

Important modern approaches to teaching focus on increasing the student's control and autonomy. Experience based learning is an aspect of constructivism that accomplishes these things, and often has a straightforward implementation. Something said by B. Ruben especially strikes home[16]:

> "If we hold too closely to the idea that effective education happens only when the learner learns what the teacher teaches, this can lead us to think that creativity is an error".

We believe that creativity cannot be taught, as much as it can be permitted, encouraged, identified, and rewarded. It is easier to be creative when the task is clear, but not restrictive, and when the domain of the task is clearly understood. An accounting system would be a terrific programming assignment if accounting were a popular hobby among sixteen to twenty year-olds.

A comparison has been done that involves business-type assignments VS game projects for intermediate programming classes [6]. The same instructor taught all classes, which involved programming in Visual Basic. The games assignments were simple board games (E.G. Monopoly), while the business assignments involved implementing a student planner or an automotive dealer vehicle purchase system. Using simple complexity measures such as lines of code (LOC), the game assignments were seen to be significantly more complex, and involved 80% more code, on the average, than the business assignments.

Two of the six constructivist assignments presented below involve programming a game. There are a number of advantages to this [5], including that games are easily understood domains, are inherently visual and event driven, can be tested easily, and are scalable. Other work has been done on the use of games as assignments [2,6].

2.3 An Objective Evaluation Strategy

Most of the assessment methods described above involve a subjective evaluation - examination results and surveys, for example. Objective results are much more difficult to obtain. One could use a standard test on the two classes, both before and after the course, and use the increase in test score as a measure of educational success. This is a trifle goal/fact oriented, but is moot in any case as no such exam was given. In any case there would be ethical issues in experimenting with a class in this manner.

The method proposed here is to use perceived effort as a measure of success. Assuming that the assignment grades in both classes are more or less equivalent, a hard thing to say objectively given that the assignments were different and were graded according to various criteria, then the assignments that involved the largest amount of work or effort should be an indicator of how much was learned. The simple argument is that more effort to achieve a similar grade implies either that more material was mastered or that the assignment engaged the students more thoroughly. This is consistent with existing practice [9,15]. In the latter case, more material was covered, whether or not it was evaluated and incorporated into the final grade. Students must learn problem solving skills, to construct new solutions for themselves.

For programming assignments in Computer Science, the source code submitted can be examined and measured for various numerical complexity values. These are objective measures of program complexity, which can be converted into measures of effort.

3 Behaviourist Model

The behaviourist approach focuses on observable behaviour, seeking to change it. In its crudest form, Behaviorism examines the way people react to a stimulus. For example, if a teacher gives a high grade, the belief is that the student is learning and doing well. If a low grade is given, then the belief is that the student is not learning.

There is an associated belief that the environment, rather than the learner, determines what is learned. The teacher's role is, therefore, to arrange the environment to elicit the desired response. The manifestation of this approach would centre on behavioural objectives, such as competency based education.

3.1 The Behaviourist Assignment Set

The following assignments are presented in the same order in which they are encountered by the students.

There are approximately 87 individuals submitting assignments in this group.

Assignment 1 - Student Grades

This requires the student to write a program that reads in percentage grades and prints corresponding letter grades.

Assignment 2 - A Point class

Create a class that represents a point in a two dimensional Cartesian coordinate system. Must have specific *set* and *get* methods, and *move*, *distance*, and *pointID*. The number of active instances are counted, and the class must be tested. The program structure is described in detail.

Assignment 3 - Class extension

Create subclasses **shape**, **rectangle**, **circle**, and **test** from the **point** class defined in assignment 2.

Assignment 4 - Mortgage Calculator

Using Swing to create a GUI, write a program to enter the number of payments, principal, and interest rate and compute results of interest such as the total monthly payment, total interest paid, and the amortization in years.

Assignment 5 - Greenhouse simulation

Simulate a greenhouse having sensors for temperature, humidity, and soil moisture. Devices (air conditioner, furnace, sprinklers) can be turned on or off. Uses threads and a GUI to simulate a period of time in the greenhouse and control of the devices.

3.2 Why are these assignments behaviourist?

It is not only the nature of the assignments that indicates behaviourism but the manner in which they are presented. In most cases the assignments are specified in great detail, giving methods, their names and parameters lists, and their functions. Very little room is given for enhancement or individual initiative.

These assignments are each designed to mimic a specific tiny application that is fully understood. The student is to replicate the instructor's solution to achieve top marks. Assignment number 5 is the nearest thing to a constructivist assignment, but even here the nature of the question - which specifies all methods, their functions, and their detailed implementation - is behaviourist.

4 Constructivist model

The constructivist approach is centred on how meaning is constructed by a student. Learning is thought to be an internal cognitive activity - students construct knowledge (models) from their classroom experience. The teacher's role is to facilitate and negotiate meaning, rather than to dictate an interpretation. The University of Calgary's recent focus on experiential learning in the undergraduate program is precisely in tune with this model.

4.1 The Constructivist Assignment Set

In each case the assignment specification describes three versions: A, B, and C. In most cases the different versions represent progressive levels of complexity: the B-version does everything the C-version does with some embellishments. This means that each increasing level involves more code (i.e. a more complex solution). Progressive levels require progressively more functionality.

This is not necessarily the case in the behaviorist set as grading is based largely on the presence or absence of specific constructs as well as a subjective assessment of that component by the marker.

Bonuses and challenges are credited differently from the main part of the assignment. This is done for several reasons. It separates the requirements for an 'A' from embellishments suggested in order to challenge the better students. It ensures that high grades remain attainable by all while encouraging excellence in those better equipped. It avoids the implication that all students should be willing or able to rise to the challenges. There were approximately 45 individuals submitting assignments in this group.

Assignment 1 -- Transition

This task allows students to familiarize themselves with the new language. They are given a small program [approximately 1-200 lines of code] in the language they already know (Pascal), and asked to re-write it in Java. Proper OO design is not emphasized. An implementation of a simple calculator is used.

Assignment 2 -- First Class

Students are to design and implement a class that serves as an enhanced version of a data type that was used in the previous program. This allows them to create a new class and incorporate it into already existing code. Specifically, they were to replace the integer type with a Big Number class that supports integers at least 15 digits long.

Assignment 3 -- Encapsulation; Simple Data Structures

Write an ACSII-graphics version of the Four Seasons Solitaire game. Emphasis is on stacks and queues; encapsulation of objects; menu-driven design.

Assignment 4 -- Parsing

Design and write a recursive parser for expressions. It must read and parse the expression, convert it into polish postfix, and then evaluate the postfix form using a stack. Parser is implemented from formal definition using BNF and syntax diagrams. Includes UML documentation.

Assignment 5 -- Inheritance

Design and implement an ASCII-graphics version of the Centipede arcade game. Turn based. Graphics implemented using 'easycurses' support class.

Assignment 6 -- Encryption in C

Implement a simple Caesar cipher in C.

4.2 Why are these assignments constructivist?

These assignments can be seen to actively engage the students, a key feature of constructivism. Based on Wheatley [18], who describes a problem-centred approach that is directly applicable, tasks (assignments) should contain the following ten attributes:

1. Be accessible to everyone at the start - all of the assignments have multiple levels so that even below average students have achievable goals.
2. Invite students to make decisions - the design is not completely specified.
3. Encourage "what if" questions - the question asked can be extended by the students.
4. Encourage students to use their own methods - rarely is the method specified in this assignment set.
5. Promote discussion and communication - the open nature of the problems encourages discussion, sometimes conducted in the labs.
6. Be replete with patterns - design patterns are discussed, but students are not limited by them.
7. Lead somewhere - the goals are clear and easily demonstrated.
8. Have an element of surprise - especially in graphical output, surprise is inevitable. Games have a random component as well.
9. Be enjoyable - the students claim it is so.

10. Be extendable - the assignment specification is essentially open ended.

It can be seen that this assignment set does satisfy all ten of the above attributes.

5 Measurement of Assignment Complexity

All of the solutions to all of the assignments were saved, and after the semester was over they were evaluated. All solutions were written in Java, and so the tools used to conduct the evaluation had to work with this language. One tool was written specifically for this work. It parsed the Java programs and collected counts of symbol usage so that the more complex metrics below could be computed. Essentially, counts of operators and operands were made so that an estimate of programming effort could be made.

What follows is a description of the metrics computed for all of the assignments from both sections of Computer Science 231. We chose the commonly encountered Halstead metrics[8], precisely because they are commonly encountered. These measures apply to systems that are working and to development efforts after coding is finished, which is certainly the case in this instance. Halstead measures first appeared in 1977 and have been the subject of experimentation and assessment ever since. They are some of the oldest measures of program complexity. These metrics are based on the simple measurements:

$\mathbf{n_1}$ = the number of distinct operators
$\mathbf{n_2}$ = the number of distinct operands
$\mathbf{N_1}$ = the total number of operators
$\mathbf{N_2}$ = the total number of operands

The remaining values below are calculated using the measurements above.

N*:* This is a measure of program length in terms of the number of tokens used by the program. It is calculated as

$$\mathbf{N = N_1 + N_2}$$

Length: The *length* is a relationship between the token length N and the vocabulary n. It is defined as:

$$\mathbf{N = n_1 \log(n_1) + n_2 \log(n_2)}$$

Vocabulary: This is the number of distinct symbols used in the definition of the program. It is defined as:

$$\mathbf{n = n_1 + n_2}$$

Lines of Code (LOC): This is a very simple measure, and is quite intuitive. As counted in the real code it is hard to decide some details. Do we count comments? Is there empty space? Obfuscated code can have very few lines.

This metric is computed from the number of tokens in the program, and presumes that there should be, on the average, 3.14 tokens per line:

$$\mathrm{LOC} = \frac{\mathrm{N}}{3.14}$$

Effort: Halstead defines effort as the total number of elementary mental discriminations. Details can be found in Halstead's book, but suffice to say that this number is an accepted measure of program difficulty.

Programming Effort is calculated as

$$\mathbf{E = V/PL}$$

where the symbol V represents a quantity named *program volume*, an estimate of the volume of information required to specify a software program; and the symbol PL is the *program level*, a measure of the relation between the volumes of the most compact representation and the actual program.

$$\mathbf{PL = 1 / ((n_1 / 2) * (N_2 / n_2))}$$

$$\mathbf{V= N * (LOG2\ n)}$$

Time to Code: This is an estimate of how long it would generally take to write the program. This measure correlates very well with the actual measured time to write programs, and is also an established measure of program difficulty or effort needed to write a particular program.

This measure is a function of the programming language use. For Fortran, the programming time T is computed as

$$\mathbf{T = E/K}$$

where the constant **K** depends on the language. For the Java language the constant 0.9 was used; this was estimated by computing the effort for a sample set of programs for which the programming time was known.

6 Evaluation of assignment sets

There were five behaviourist assignments to be completed in the winter semester of 2002, during which time six constructivist assignments were completed. In most cases we compute the mean value of a metric per assignment, as well as the total for the semester.

The most simple measures of complexity deal with simply the number of tokens, of various sorts. Mostly the number of operands is the deciding factor, increasing the complexity of the constructivist set to the point where it has almost twice the measured complexity of the other set. Here is a summary of what was measured in the actual student assignments, averaged over all assignments.

Number of tokens N: Constructivist = 481.1, Behaviourist = 368.6.
Vocabulary: Constructivist=533.8, Behaviourist=315.4.
Length: Constructivist=4846.4 (total=***29,078.5***), Behaviourist=2556.2 (total=**12,780.8**).
LOC: Constructivist=153.2 (total=***919.3***), Behaviourist=117.4 (total=**586.9**).

The other, derivative, complexity measures tell a definitive tale.

	Effort (Median)	
Assignment	Constructivist	Behaviourist
1	9178.4	15946
2	18316	10102
3	29045	7669.9
4	22018	11427
5	40438	22972
6	30466	
Unbiased mean	***24,910***	**13,623**

Total Effort	***149,461***	**68,117**

Time (to code, median)

Assignment	Constructivist	Behaviourist
1	2.8328	4.9215
2	5.6531	3.1178
3	8.9647	2.3672
4	6.7956	3.527
5	12.481	7.0902
6	9.403	
Unbiased mean	***7.69***	**4.20**
Total Time	***46.13***	**21.02**

We think of the metrics above as measures of work done, or effort. Again, the constructivist set is nearly double the effort of the behaviourist set on a per assignment basis. It is more than twice the effort overall during the semester.

7 Conclusions

The constructivist assignment set appears to be, on the average, about twice as much effort as is the behaviourist set. We would expect that the students who completed this collection would be better prepared for subsequent programming courses, a hypothesis that we propose to test over the next few years. Specifically, in the case of assignment 1 the behaviourist assignment is more complex than the constructivist. This is bourn out by all of the other measures evaluated.

All of the other metrics favour the constructivist set as being more complex and requiring more effort. What will be attempted next is to follow the performance of the two groups of students through the next year, to try to see whether there is a trend in their performance. Does the completion of either of these assignment sets predict a superior performance in future courses?

8 Acknowledgments

We thank D. Walters at the U of C Computer Science Department for conducting the measurements on the assignments, and K. Barker and L. Manzara for providing resources and data.

9 References

[1] D. H. Andrews and L. A. Goodson, A Comparative Analysis of Models of Instructional Design, *Journal of Instructional Development*. 3:(4) , 1980. Pp 2-16.

[2] K. Becker, Teaching With Games: The Asteroids! and Minesweeper Experience, *Journal of Computing in Small Colleges*, Vol. 17 No. 2, December, 2001. Pp. 22-32.

[3] M. Ben-Ari, Constructivism in Computer Science Education, *Journal of Computers in Mathematics and Science Teaching*, 20(1), 2001. Pp. 45-73.

[4] R. Crawford, Teaching and Learning IT in English State Secondary Schools - Towards a New Pedagogy, *Education and Information Technologies*, 4, 1999. Pp 49-63.

[5] B. Dobing and D. Erbach, Building Games as Programming Projects, *Proc. 14 Annual Conference of the International Academy for Information Management*, Charlotte, N.C., 1999. Pp. 298-302.

[6] B. Dobing and D. Erbach, A Comparison of Business and Game Projects for the Intermediate Programming Classes, *Proc. 16 Annual Conference of the International Academy for Information Management*, New Orleans, LA., 2001. Pp. 287-296.

[7] Valerio Franceschin, Complexity -- Software Metrics, **http://sern.ucalgary.ca/Courses/cpsc/451/W02/Complexity.html**

[8] M. H. Halstead, *Elements of Software Science*, New York, Elsevier North-Holland, 1977.

[9] D. L. Kirkpatrick, Techniques for Evaluating Training Programs, *Training and Development Journal*, June, 1979. Pp 178-192.

[10]K. C. Lee, JavaNCSS - A Source Measurement Suite for Java, **http://www.kclee.com/clemens/java/javancss/**

[11]R.K. Lind and K. Vairavan, An Experimental Investigation of Software Metrics and their Relationship to Software Development Effort, *IEEE Transactions on Software Engineering*, v15, p649(5), May 1989.

[12]McCabe, Complexity Measure, *IEEE Transacions on Software Engineering*, Volume 2, No 4, pp 308-320, December 1976.

[13]S.B. Merriam and R.S. Caffarella, *Learning in Adulthood: A Comprehensive Guide* (2nd Ed), Jossey-Bass, San Francisco, 1999.

[14]G. Miller, The Magical Number 7 Plus or Minus Two: Some Limits on Our Capacity for Processing Information, *Psychological Review*, 63., 1957. Pp. 81-97.

[15]R. Romer, Making Quality Count in Undergraduate Education, Denver Education Commission of the States, 1995.

[16]B. D. Ruben, Simulations, Games, and Experience-Based Learning: the Quest for a New Paradigm for Teaching and Learning, *Simulation and Gaming*, Vol 30, No. 4, Dec, 1999. Pp. 498-505.

[17]E. Soloway, Learning to Program = Learning to Construct Mechanisms and Explanations, *Communications of the ACM*, Vol. 29, No. 9, Sept. 1986. Pp 850-858.

[18]G. H. Wheatley, Constructivist perspectives on science and mathematics learning. *Science Education* 75 (1), 1991. Pp. 9-21.