IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP5240

This paper appears in *Managing Modern Organizations Through Information Technology*, Proceedings of the 2005 Information Resources Management Association International Conference, edited by Mehdi Khosrow-Pour.

Teaching Programming: The Shift from Conventional to Student-Centred Approach

Matthew Butler

School of Multimedia Systems, Monash University, Clyde Rd., Berwick VIC 3806, Australia, matthew.butler@infotech.monash.edu.au

ABSTRACT

Educators are under constant pressure to update and adapt both their teaching materials and methods, and for today's teachers the focus is on creating curriculum that is student-centred. Although almost all educators will acknowledge the necessity to put the needs of the students at the forefront, as well as to tailor delivery based on the individual requirements of the students, there is a tendency to believe that all curriculum must have this focus. This paper will suggest that this should not be the case, and that curriculum development for Programming subjects must begin in a more "traditional" approach, before blossoming into a student-centred, constructivist model.

INTRODUCTION

University educators are under constant pressure to update and adapt both their teaching materials and methods. There is an obvious need to revise curriculum based on conceptual and technological advancements within the discipline, as well as societal shifts in the disciplines place in the larger context. However the other element educators must take into account is the continual evolution of the students in question.

Because of the disparate groups that exist within today's student cohort, notions of "Student-centredness" and "Student-centred learning" have become prevalent in any discussion of tertiary curriculum development. They are concepts that are now inherently expected to be adopted as *the only* way to develop and deliver course material. As Gosling points out in "A Handbook for Teaching and Learning in Higher Education", "Higher education no longer operates entirely on a teacher-centred model of teaching and is shifting, albeit slowly and hesitantly, towards a more student-centred model" (Fry et. al 2003, p. 163).

Although few educators will argue this as an overall philosophy, regardless of their penchant for Objectivist or Constructivist ideals, there appears to be a tendency to believe that *all* content development and delivery must address the notions of student-centredness. The author argues that although they agree with the fundamental principles of this approach, there is also a need at times to provide a very focused, explicit, delivery of core material, particularly in certain disciplines.

Using the learning and teaching of Programming as a basis, the author will demonstrate that a successful student-centred approach to teaching can at times stem only from a reasonably rigid (or "conventional") beginning to the subject. This case study will also highlight a need in the development of Programming curriculum to not just consider individual units in isolation, but to develop an entire structure, from introductory level to advanced, to ensure that student needs are adequately addressed.

THE NATURE OF LEARNING PROGRAMMING

In order to use the teaching and learning of Programming as a case study, we must first acknowledge the curriculum of the programming context. To begin, the study of Programming Languages is, in general, the learning of a particular Programming Language, be it Pascal, Visual Basic, Java, or any other language. Martin (1996) and numerous other authors suggest there are two distinct skills being developed in the programmer:

- Problem solving, or logic skills
- Knowledge of a programming language

In order to develop these skills, the study of a programming language will usually encompass:

- Background, principles, and general application of the language in question
- Learning of the Programming Language syntax
- Specific development processes for the language
- Troubleshooting and testing techniques
- Creating programs using the language in question

For novice students, becoming familiar with programming principles is the key to being able to embrace a programming language for the first time. For the mathematically minded, the key concepts and structures behind programming (such as the notions of sequence, selection, and repetition) are a natural extension of the logical thought processes they have developed in this other discipline. But for many, the mindset required for programming is one that needs to be introduced and nurtured. Broad conceptual ideas are somewhat replaced with strict methodologies.

The learning of the programming language syntax is obviously of key importance, and similarly the focus on syntactic specifics depends on the experience of the student. For those with no prior knowledge of the language, the learning of the syntax can be a slow process that involves a methodical approach of introducing new elements coupled with significant basic application. For the veteran, more sophisticated application of the language is the key, although this is still often coupled with the introduction of new syntax. Where specific focus is placed in the curriculum depends on the level of the subject. An introductory unit will generally have more focus placed on the syntax of the language, while at a higher level, where a knowledge of the bulk of the syntax has been obtained previously, greater emphasis will be placed on learning more specialised syntax, whilst primarily creating more sophisticated programs using the language to its fullest capacities.

How does this differ to other disciplines? There is obviously a greater focus on logical processing, troubleshooting, and general problem solving. As Doube (1998) highlights 'Introductory computer programming subjects can be highly abstract and conceptual. They often aim to promote the application of sound design and problem solving strategies…' (p. 86).

Aside from the obvious conceptual differences, within the Programming context there is a need to be introduced to core programming concepts as well as key syntax, before the student can perform any independent application of concepts. Whereas many other disciplines encourage an exploration of knowledge and concepts from day one, because the concepts may have some basis in everyday experience, programming is

a discipline that is often very new to many students. Logical and mathematical problem solving techniques underpin programming, and it is often that a student has not developed or applied these concepts previously… at least in a similar context. As a result, there is a period of familiarisation and understanding of these concepts that must take place.

The other obvious difference is in the need to learn and understand the programming language itself. Although many other disciplines have a lexicon that is required for meaningful conversation in the area, a student can usually express their ideas in some way before becoming familiar with the explicit language. For the programming student however, they cannot "converse" in the programming environment until they have a basic grasp of the syntax and semantics of the language.

LEARNING THEORIES IN THE PROGRAMMING CONTEXT

Taking this into account, on what end of the teaching philosophy spectrum does the teaching of programming languages exist? This can be considered from a number of different perspectives, primarily depending on the base theoretical model chosen, which as a result makes it is difficult to espouse a definitive learning theory that applies resolutely to the programming context. One can discuss the relative merits of taking an objectivist approach, and likewise a constructivist mentality can be validly argued. In fact there are times when both may have a place in programming curricula. Leidner and Jarvenpaa (1995) state resoundingly that "No particular model is the best approach" (p. 271).

Similarly, Eijkman discusses the work of Gundy, noting "three types of human interests, which influence how knowledge is generated and organised; the technical, practical and transformative". On face value the technical perspective may seem most appropriate; an orientation that "reflects a distinctly technical, product-centred interest" (Eijkman 2004). However although for a programmer the end product is of the utmost importance (and indeed in industry, is the journey rewarded?), it could be argued that for the programming beginner, the importance must be on developing sound methodological processes within the student. Thus programming curriculum can become more akin to the practical interest perspective, where students must explore their environment, explore domains of knowledge, and make mistakes in the process of gaining a deeper understanding of the discipline.

The work of Kolb also underscores that these apparent conflicts exist. Healey and Jenkins (2000), in their discussion of Kolb's work highlight that "Some of the theory's appeal is that it provides a rationale for a variety of learning methods" (p. 186)

A more lateral approach to the application of learning theory to the programming context may be to move away from approaches traditionally aligned with technical disciplines, and to draw comparison to the learning of spoken languages. This also becomes more likely when we consider that the metaphors of "conversation" and "dialog" are now commonplace within learning theory. For example, Applebee (1996) suggests "A curriculum provides domains for conversation, and the conversations that take place within those domains are the primary means of teaching and learning" (p. 36).

A programming language, just like a spoken language, has not only a syntactic element, of learning the actual "words", but also a sematic component, where the words must be placed together to have meaning. Semantic understanding is a skill often developed in the introductory units mentioned previously. It is of key importance that skills in this area are developed as early as possible in the student to ensure success both in the immediate language being learned, and also in the study of future languages.

SHIFT FROM CONVENTIONAL TO STUDENT-CENTRED

The discussion of the previous section alludes to the fact that just like teaching in general, no one learning theory stands above the rest in the programming context. It is clear that there is a large element that revolves around the development of skills within practical application of concepts learned. As the ethos of student-centred learning and constructivist theory espouses, "Instruction must be concerned with the experiences and contexts" (Kearsley, http://tip.psychology.org), where students are required to make choices about what and how they learn. However, can this exploration take place at an introductory programming level without a strictly guided grounding in core concept? Can fundamental core concepts be obtained by the student if they are given the ability to freely roam the discipline from the beginning?

It is easy to dismiss the conventional approach as being an antiquated approach to teaching, however based on the discussion above, it does have merit. The conventional approach suggests that decisions regarding curriculum and delivery are made by the teacher, whereas in student-centred learning, the student is the one who makes choices about what and how to learn. How does one make the decision what to learn if there is no grounding yet in the fundamentals of a discipline? In order to facilitate student-centred learning, the student must have a fundamental grounding in the area of study, so they themselves can decide what is most important. If the teacher has not provided the student a grounding in the fundamentals of programming for example, the student cannot make choices about the direction their study of programming should go, or in what contexts they can apply it.

To many, programming is a new discipline and indeed a new language that must be learned. Therefore as Applebee (1996) points out "Exploring a topic in science, for example, involves using different rules of argument and evidence than will be used in mathematics, and both will differ from history" (p. 37), we cannot expect an introductory student to be cognizant in the domain of programming without initial instruction.

Lemke (1994) strongly argues against any form of curriculum structure, as far as hoping that "… the dominance of the Curricular Model of education will be destroyed once and for all…" (p. 2). But even Lemke acknowledges that "It is obviously true that the very youngest humans need to learn the basic tools of the natural language and the cultural categories of the society into which they are born in order to function…" (p. 3). Although loathe to equate an introductory programming student with a child, in many cases the programming student is in a similar position. How do they function in an environment such as programming with no frame of reference or formal introduction?

Indeed at a more advanced level students have the ability, and should indeed be encouraged, to explore domains of knowledge. With an understanding of the core concepts, basic syntax and its semantic understanding, it must be up to the student to apply this to their own frames of reference (in this case application of the language) and to take it into new areas. The educator must be a facilitator in the student learning new syntax and application.

Several learning theories concur with this view, if indirectly. Cross discusses the characteristics of Adult as Learners, and although it can be argued that the definition of adults is different in this context, elements are still relevant. Cross argues that personal and situational characteristics are variables in approaches to teaching and learning. If we consider the previous contrast of introductory level students and advanced student, then it is clear that both the type of student and the situation they are in varies, so as a result we cannot pigeon-hole an overall programming curriculum into being "traditional" or "student-centred". In fact Fenwick and Tennant (2004) in their discussion of Adult Learners state that "… there is no one best way to understand learning, just as learners and educators are each very different and constantly changing" (p. 55).

Likewise, many of the learning theories that exist have context as a key factor or variable. Kearsley (2004) points out that Spiro et al assert "effective learning is context-dependant", and although this means that all teaching must be context-driven, it also infers that effective learning must be tailored to the appropriate context, meaning that the context for a first year is very different to that of a third year student. Do introductory students even have a legitimate context?

PROGRAMMING CURRICULUM DEVELOPMENT

What this discussion of both programming subjects in general and the relevant learning theories highlights is that there is a need to consider the overall learning process involved with a programming language and therefore acknowledge that as curriculum developers we cannot focus solely on one or a handful of likeminded learning theories. We must embrace seemingly disparate theories and ideas.

Healey and Jenkins (2000) have already embraced this way of thinking. As they point out "There is also evidence that learning styles are related to the stage students are in their studies. Nulty and Barrett (1986) found that students in their first third of studies adopted learning styles that were similar to each other irrespective of main disciplines. However, the learning styles of students in the final third of their studies tended to be related to the discipline that was the primary focus of their studies." (p. 189). This ideal must be incorporated into the development of Programming curriculum.

Introductory Visual Basic .Net (VB .Net), as taught within the School of Multimedia Systems at Monash University, is for many students their first introduction to programming. As a consequence, the unit covers not only learning of VB .Net for the first time, but programming concepts in general. Curriculum for the unit therefore is a balancing act of introducing language syntax, language semantics, programming concepts (such as variables, programming structures, modularity), and methodology.

As discussed previously, how can a student, exposed to programming for the first time, be expected to self-explore programming domains with any confidence? How can a student not be intimidated by crashing their computer with an infinite loop, if they have no idea what has happened? For these reasons, students deserve a well structured introduction to the language, one where they are guided into the "domain of knowledge".

This is not to say that the introductory levels must be devoid of any constructivist ideals. Indeed, exploration within the tutorial setting is essential to a deeper understanding of material. But this cannot be thrust upon the student from day one… development must be nurtured. If the introductory unit of VB .Net is considered to be a springboard to further programming study, be it in VB or any other language, then this structure is important.

As the study of the programming language progresses however, a distinct shift in approach is required… a shift toward constructivist, or student centred ideals. Once the student has a grounding in programming ideals as well as the syntax, then the student can deepen their understanding of the material. Bruner's theory says that "the instructor should try and encourage students to discover principles by themselves" (Kearsley, http://tip.psychology.org/bruner.htm). Although this may seem at odds with earlier discussion, particularly relating to the need to dictate fairly explicitly what the students must learn, it can be said that this is to introduce the concepts and provide a starting point for investigation… they are indeed discovered and learned by the students with practical application in tutorials and assignments.

This is not to suggest that students studying advanced programming have free rein to wander the programming domain, clear learning objectives are still needed. However the individual learning style of the student can come to the fore, along with individual learning desires. This can be done as both student and teacher can have comfort that the learning will not be without purpose or grounding in fundamental principles. Exploration can also become a more lateral process, yet still founded in fundamental understanding.

So to ensure that the programming student is provided with the most thorough and effective learning environment, the development of a programming curriculum must consider both the introductory and advanced context. Focus on solely an introductory level can deprive the student of the ability to explore their new-found knowledge, providing a valid argument against the objectivist approach. Similarly curriculum for advanced concept cannot be considered in isolation. We must ensure that students are not left to flounder with concepts that are completely foreign, just for the sake of providing a constructivist approach, and creating a "student-centred" environment… one that does not address student needs after all.

CONCLUSION

It is unfair to disregard an objectivist mode of teaching, just because it does not conform to current learning theory popularity or trend. At the introductory level, a structured delivery of programming concept and theory, programming structures, and specific syntax and semantic, must be provided. Rather than allow students to wistfully find these concepts for themselves, in the interest of both the students and providing a framework for exploratory learning, the initial stages of programming curriculum should conform to objectivist, or traditional, principles.

De Vita (2001) suggests that "Good practice must, therefore, translate into using a variety of teaching styles and address each side of each learning dimension at least some of the time" (p. 170). This is a fundamental statement that underpins this discussion. As academics, particularly in today's teaching environment, it can be too easy to be under the impression that all of our teaching must conform to entirely "student-centred" approaches. Although it is essential that all our teaching is indeed focused on the needs of the student, we cannot ignore the need to provide basic guidance into the domains of knowledge.

As a result, in looking at the development of programming curriculum, academics must consider the entire learning path of the student. Indeed culminating in constructivist ideals, having the students dictate learning approaches and directions, we must ensure that we address a central question of Applebee (1996), that of "If students are to learn by doing, how do schools avoid allowing them to wallow in their own ignorance?" (p. 36). By providing some structured guidance in the beginning, the final destination can be reached without this concern.

REFERENCES

Applebee, A. N (1996), Chapter 4: "Curriculum as Conversation", *Curriculum as Conversation: Transforming Traditions of Teaching and Learning*, Chicago: University of Chicago Press

De Vita, G (2001), "Learning Styles, Culture and Inclusive Instruction in the Multicultural Classroom: A Business and Management Perspective", *Innovations in Education and Teaching International*, 38(2), p. 165 - 174

Doube, W. (1998), "Multimedia delivery of computer programming subjects: basing structure on instructional design", *The proceedings of the third Australasian conference on Computer science education*, The University of Queensland, Australia

Eijkman, H. (2004) "Curriculum as social construction: three orientations to practice", Monash University, HEDU

Fenwick, T. and Tennant, M. (2004), "Understanding Adult Learners", in Foley G. (ed) *Dimensions of Adult Learning*, Allen & Unwin: Crows Nest NSW

Fry, H. Ketteridge, S. and Marshall, S. (2003) "A Handbook for Teaching and Learning in Higher Education", 2nd Edition, RoutledgeFalmer

Healey, M. and Jenkins, A. (2000) "Kolb's Experiental Learning Theory and its Application in Geography in Higher Education", *Journal of Geography*, September/October 2000; 99(5) ProQuest Education Journals, p. 185

Kearsley, G (2004), "Explorations in Learning & Instruction: The Theory Into Practice Database", http://tip.psychology.org/ (accessed 06/06/2004)

Leidner, D. E and Jarvenpaa, S. (1995) "The Use of Information Technology to Enhance Management School Education: A Theoretical View", *MIS Quarterly / September 1995*, p. 265 - 291

Lemke, J.L (1994) "The Coming Paradigm Wars in Education: Curriculum Vs Information Access", http://academic.brooklyn.cuny.edu/education/jlemke/papers/cfppaper.htm (accessed 02/06/2004)

Martin, J. L. 1996, 'Is Turing a better language for teaching programming than Pascal?', http://www.holsoft.com/turing/essay.html, (accessed 23/08/2002)