Enhancing Student Engagement in an Introductory Programming Course

Arnold N. Pears,
IT Department, Uppsala University, Box 337, 751 05 Uppsala, SWEDEN,
arold.pears@it.uu.se

Abstract—Adopting a scholarly approach to teaching and learning and the instructional design of tertiary courses has gained increasing popularity in recent years.

A programming course case study is used to demonstrate how research results on the teaching and learning of programming, can be used to inform instructional design of an introductory programming course.

I. INTRODUCTION

The complexity of approaching teaching and learning in a discipline requires tools and access to an education research discourse. While academics are familiar with the technical research in their own area of science or engineering, their awareness of research into the teaching and learning of their discipline is often limited. Even when academics are aware of the relevant educational discourses within their discipline, applying the results to practice remains potentially problematic.

There are few examples of curriculum design relate relevant educational and teaching and learning reseach to the design and implementation of CS courses. This paper attempts to fill part of that gap by describing a research based course design approach.

II. CASE STUDY

A. Guiding Principles

Much of Computer Science teaching embraces a constructivist epistemology. Based in the constructivist tradition Our case study course design aims to encourage student-centric learning and draws on educational research on motivation and activated learners [1], deep vs surface [2], [3] approaches to learning activities, and theories explaining how students transform their understanding (such as threshold concepts [4] and research on conceptual change [5]). Given the course topic we also need to draw on CER literature that characterises the activity of programming and what it takes to learn to program [6]–[8].

Hands on experience was used throughout the course to build up student familiarity with language structures and concepts. The importance of this for the construction of programming knowledge in computing is emphasised by Eckerdal [9] There is a huge debate in computing about how to teach object oriented programming skills, and when to do this in a university programming course sequence [10]–[12]. Here there is no approach which research demonstrates to be more effective [13], so a procedural and control structures approach using Java was adopted after consultation with the degree programme coordinator.

B. Motivation and Deep Learning

The role of creativity as a motivator for both university and high school students learning to program is analysed with promising results by Romeike and Knobelsdorf [14]. This work implies that it is important to provide opportunities for programming creativity in both laboratory programming exercises, and in larger scale programming exercises. This is also emphasised by Malmi et al. in a recent investigation of student attitudes towards programming practical exercises [15].

Motivation is addressed by talking to students about what they want to achieve and relating this to the content and structure of the programming to be undertaken. Students were encouraged to define their own projects within certain constraints. Definition of individual programming goals (through a reflective exercise) helps individuals to enunciate personal goals, and reflect on their learning. Reflection and engagement are strongly linked to predisposing students to adopt deep learning tendencies [16], [17]. We also adopt practices based on the research of Barker et al. [18] designed to promote an open and constructive class communication culture, which has also been linked to improved learning outcomes for CS students.

C. Self Efficacy and Self Theories

Student beliefs about their own ability to successfully engage with the topic matter to be learned can also influence what they learn and how successfully they learn it [1].

Peer programming, governed by strict instructions to reason aloud, and reminders to swap driver/navigator roles regularly. This helps students to self calibrate in terms of their perceptions of their own competence in dealing with the material to be learned and their recognition of personal achievement. Research has also demonstrated that this encourages peer learning and promotes improved outcomes in a programming student cohort [19]–[21].

D. Making Tacit Knowledge Explicit

Lectures were designed to meet student needs and crafted to help students to discern facets of the "expert tacit knowledge" associated with software development. Like experts in many other areas the techniques expert programers use to develop code and correct errors are often tacit [22].
teachniques and skills are certainly not visible in the normal lecture environment where pre-tested code is presented in a sequential manner, or in whole blocks on an overhead slide. We concluded that available research suggested that providing learners with an opportunity to observe expert practice played an important role in assisting them in quickly acquiring a sophisticated approach to code development and debugging.

Students were also provided with a large body of code examples, both our own and via online tutorials and repositories. This rich resource pool provides good “exemplars”, creating a context for discerning what constitutes “good code” and providing resources and support for exploring program behaviour in a visualisation environment. To support development of an intuitive understanding of the von Neumann machine model we used Jeliot [23].

E. The Role of Practice

The reinforcement of theory through practice is achieved by interspersing lecture and discussion presentations with hands-on implementation and code exploration exercises. A ten-week introductory phase consisted of one 90 minute full class interactive programming demonstration by the instructor, followed by a 90 minute coding laboratory and additional homework exercises.

A further ten weeks were devoted to group projects. Programming groups were given a week to develop a project proposal (based on generic instructions), which they then presented to the lecturer and the class. If the project was sufficiently complex, involved at least two classes, and stored and retrieved complex information in a primitive database, it was approved, and the group began implementation. During the ten project weeks the 90 minute laboratory sessions were used by the lecturer to observe student’s progress. Groups were required to attend and demonstrate their code regularly to enable the lecturer to take notes on the learning progress of individuals. These notes, together with a final group and individual interview of 30 minutes and 10 minutes respectively provided the data used to determine the final grade of each student.

III. Conclusions

Scholarship of Teaching and Learning involves applying scholarly values to academic’s educational activities. This paper demonstrates how an instructional design can draw on disciplinary teaching and learning scholarship, and educational research. The main contribution of the paper is the exemplification of an approach to managing the transfer of research results into practice, and a discussion of the key aspects of this process in the context of teaching introductory programming.

REFERENCES


978-1-4244-6262-9/10/$26.00 ©2010 IEEE 40th ASEE/IEEE Frontiers in Education Conference

F1E-2

Session F1E