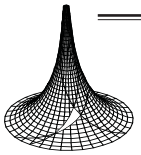

2/2 (2004), 407–421
tmcs@math.klte.hu
http://tmcs.math.klte.hu

Teaching
Mathematics and
Computer Science

Mechanisms for teaching introductory programming using active learning

Michael Wirth

Abstract. One of the requirements of teaching introductory programming to students whose branch of learning is engineering or science is bridging the gap between in-class lectures and real-world applications. Traditional passive approaches to lecturing often focus on the syntax of a language with little or no discussion of the process involved in using the language to design algorithms to solve real-world problems. One way of overcoming the limitations of traditional lecturing is by tailoring lectures towards becoming more student-oriented, a pedagogical methodology known as active learning. This paper explores mechanisms for implementing active learning in introductory programming courses in computer science.

Key words and phrases: introductory programming, active learning, student interaction, classroom teaching.

ZDM Subject Classification: C70, D40.

1. Introduction

Teaching introductory programming to students with little background in computing can be a challenging task, which is made more difficult when students are from diverse disciplines. Traditional methods of teaching such classes using passive instruction are inadequate in providing a comprehensive learning environment. Often the challenge of teaching programming to such a diverse audience is bridging the gap between in-class lectures and real-world applications. One way

of overcoming the limitations of traditional lecturing is by tailoring lectures towards becoming more student-oriented, a pedagogical philosophy where students assume a participatory role in the lecture.

This paper explores the benefit of active learning over traditional passive methods in multidisciplinary introductory programming classes, and describes some of the mechanisms used for implementing active learning. The anecdotes in this paper come from the biannual offering of "Introduction to Programming", at the University of Guelph. The course is taken by 300–500 students composed of both CS majors and students from engineering and science. The syllabus is comprised of the foundations of programming in C (data types, loops, decision statements, arrays, functions).

2. Limitations of lecturing

When teaching the foundations of programming, much of the content traditionally deals with conveying the elementary notion of programming language syntax [1]. In one sense, programming languages are no different from spoken languages. Spoken languages are composed of words and linguistic features and have certain natural "rules" that determine what passes for sensible communication. These rules govern how words conveying messages are combined in a language to form meaningful sentences. Similarly, programming language syntax describes the structure of language elements. In both circumstances knowledge of the syntax is of little use without a realization of the process involved in applying the language.

Lecturing is the most common method of teaching computer science, yet it is often the least effective way to teach the process of programming [2]. Indeed, Laurillard [3] describes the process of lecturing itself has been referred to as "a grossly inefficient way of engaging academic knowledge". It yields students who can memorize language syntax, but have difficulty in applying knowledge about how this syntax can be used in designing algorithms. This is partially a result of lectures teaching content at a relatively low level of learning – imparting facts, principles, theories, terminology, symbols, and other knowledge information. Traditional passive methods of lecturing customarily rely on a notion of learning known as "memorizing and reproducing" [4]. Here learning is entirely related to the anticipated reproduction of what is learned, to some educational control or assessment [4]. This implies that the students knowledge is not enhanced, they

merely prove they have learned the material by being able to reproduce it. Passive learning relies on the student to absorb knowledge, with little recourse to ask questions and clarify points. A typical passive lecture results in long periods of uninterrupted instructor-centered expository discourse that regulates students to the role of passive spectators in the classroom. Students often spend most of the time writing notes with little time to reflect on the material presented in the lecture.

While lecturing can be effective, for instance in providing an opportunity for a large number of students to be simultaneously exposed to a new topic, it labors under a number of inherent limitations. Firstly, students' attention to what the instructor is saying (i.e. their ability to concentrate) decreases as the lecture proceeds [5]. This lack of attention manifests itself in a reduction in the amount of information retained by the student. For the first 5-10 minutes of a typical 50 minute lecture a student remembers a high proportion of the information presented, after which the proportion of information preserved rapidly declines [6]. Students typically retain 70% from the first 10 minutes of lecture, and 20% from the last 10 minutes [7]. The average percentage of material retained in long-term memory is also influenced by the mode of interaction used in the lecture [8]. Whilst learning by "seeing and hearing" results in a 50% retention rate, active learning which involves "doing and discussing" results in 90% retention. A number of factors have also been identified that have a negative impact on memory [6]. One of these relates to interference. Passive lecturing often occurs as an uninterrupted stream of information, and what comes before or after a piece of information usually interferes in a negative manner with the information. This phenomenon is called the bowing effect because the information in the middle is affected by both pro- and retro-active interference [9].

Bloom [10] lists a progression of learning from simple to complex: knowledge, comprehension, application, analysis, synthesis and evaluation. Passive learning only encompasses the first of these paradigms, mostly notably knowledge (observation and recall of information), and as such is not well suited to higher levels of learning. A comprehensive review of the effectiveness of traditional lecturing compared with other techniques was conducted by Bligh [9] in which he concluded that lectures were approximately equivalent to other methods for acquiring knowledge, but were often less than effective in promoting thought, changing beliefs and developing analytical skills. In summary, passive lecturing, in the context of teaching programming languages, is at best useful in conveying

a sense of the basic syntax of a programming language, it does not relate the process for using the language to design algorithms to solve real-world problems.

3. The role of active learning

Interaction in the classroom seldom happens by chance. Simply posing questions to the tune of "Does everybody understand this?" usually results in silence, partially because students have spent the entire lecture writing down material, with little opportunity to reflect on the content. Taylor [11] states that:

> *"teaching begins with the premise that if I want students to become more effective in meaningful learning and thinking, they need to spend more time in active, meaningful learning and thinking – not just sitting and passively receiving information"*

According to social psychological theories, learning is more effective when process is an active rather than a passive one [12]. The process of having students engage in some activity that forces them to think about and comment on the information presented is known as active learning. Active learning is an umbrella term for a variety of educational approaches focused on student-oriented learning. It is sometimes used interchangeably with terms like collaborative learning or cooperative learning, although both are essentially subsets of active learning. Active learning requires students to take a participatory role in learning, rather than adopt a more passive position [13]. Active learning activities vary considerably, but most focus on students' exploration or application of knowledge, not simply the instructor's presentation or explication of it. It can encompass a range of activities, including problem-based learning, case studies, simulations, workshops, and discussion groups.

Active learning involves more than just the notion of student interaction. In order to better facilitate these activities, the lecture itself must be augmented using mediums such as electronic lecture notes [14]. Electronic lecture notes are a concise electronic medium designed to provide a simple introduction to key concepts, including copies of any detailed figures and programming examples. The goal of electronic lecture notes is to support active learning by drawing the attention of the student away from rote note-taking to focus more on the perception, comprehension [15] and analysis of knowledge. To facilitate active learning students must adopt higher order thinking [16], employ critical thinking skills – the

ability to apply and analyze the knowledge. This is especially pertinent in computer science where problem resolution requires defining a series of algorithms, and implementing these algorithms in a particular programming language. Active learning has been shown to increase learning in computer science courses [1,17] and can be achieved in a number of ways.

3.1. Refocusing attention/fostering motivation

Compared with the uninterrupted lecture, variations in teaching methods will usually result in higher levels of attention. Indeed a pause triggered by the change in focus from a lecture to an interactive session will allow the level of attention to recover. The provision of problems, case studies, interactive discussions and ways of involving students fosters motivation and influences the way in which students perceive knowledge.

3.2. Helping with the retention of information

Bligh [9] notes that information learned before a pause is better remembered, implying that a lecture benefits from pauses, or a change in focus.

3.3. Reinforcing understanding

The use of active learning techniques such as those discussed serves to promote a deep approach to learning, rather than the more surface approach often encountered in introductory programming classes. A surface learning approach focuses on memorizing facts and principles, without much thought – e.g. learning the syntax of a programming language without putting it into a broader context or seeking an understanding of how it can be applied. A deep learning approach emphasizes thought rather than memory, focusing on meaning and understanding, – e.g. comprehension of programming syntax through its use in case studies.

3.4. Encouraging critical thinking and analysis

One of the cornerstones of active learning is the notion of students becoming actively involved in the learning process, placing less emphasis on memorization and more emphasis on critical thinking and problem solving. Rather than passive acceptance of prescribed ideas, students are encouraged to realize and develop

their own beliefs. Active learning allows students to progress in Bloom's taxonomy [10] beyond the mere recall of knowledge. Questions such as "What is the difference between loops and recursion?" require analytical thinking, whilst questions like "What is the benefit of using pointers to define arrays?" requires more evaluative thinking. Some sample questions illustrating Bloom's taxonomy in the context of programming are shown below:

Bloom's Level Sample Questions

Knowledge:	Name the three looping structures in C?
Comprehension:	What is the purpose of recursion? Describe how it be used to design an algorithm. Differentiate between recursion and iterative programming?
Application:	Describe the effects of passing an array to a function? Illustrate how recursion could be used to calculate the Fibonacci series.
Analysis:	Compare the differences between a iteration and recursion.
Synthesis:	Design an algorithm to model the population dynamics of predators and their prey.
Evaluation:	Explain how the integer overflow bug which occurred during the launch of the Ariane 5 rocket could have been avoided. Provide a rationale for your answer.

4. Active Learning Mechanisms

To facilitate active learning in our introductory programming classes we use five basic mechanisms to increase active participation, both between students and the instructor and amongst students themselves. There is no firm methodology for when to incorporate specific mechanisms, but the basic format of a lecture consists of one of the following lecture formats interspersed with learning activities to refocus the students attention:

- A *feedback* lecture – two mini lectures separated by a small-group exercise.
- A *guided* lecture – a half-class lecture, followed by a small-group activity: e.g. using a case study to illustrate a programming construct.
- A *responsive* lecture – an open-ended lecture devoted to answering student-generated questions. Questions may or may not be submitted in advance.

4.1. Insight puzzles

One way of increasing the amount of class participation during lectures is to have a problem solving session before the start of every class, as the students are arriving. This involves using "insight puzzles" to facilitate lateral thinking. These problems are in some way illogical, improbable, or even contradictory. Breakthroughs in thinking are made by recognizing subtle clues, challenging assumptions, and seeing old situations in new ways. This teaches students how to think in nonlinear ways, question or check assumptions, eliminate irrelevant information, and distinguish between causal, corollational, and coincidental relationships. How does it work? The puzzle is displayed on the screen and the class may ask questions. Students that already know the answer are asked to keep quiet and not spoil the learning experience of others. Students learn to ask questions of relevance, clarifying questions, and questions that help eliminate options, that are useful in solving the puzzle. Students learn to use logic, check assumptions and examine situations from multiple perspectives. This routine accomplishes several ends: It promotes creative thinking, and adds some dynamism to the class at the outset of each meeting, acting as a catalyst to further class participation. It also helps encourage higher levels of thinking, developing intellectual skills that can benefit students beyond the material covered in class. An example of an insight puzzle is described below:

The Equation

Here is an equation: $2+7-118=129$. As it stands, it is not a valid mathematical statement. Add one straight line anywhere in the equation to make it a true statement. There are at least three solutions:

- Put a slash through the equals sign to make it "does not equal".
- Put a diagonal line upward from the right end of the equal sign to make the expression read "less than or equal to".
- The + can be changed to a 4 by adding a vertical line on the upper left of the sign. This makes the equation true.

4.2. Questioning and discussion sessions

One of the goals of active learning is to entice students into participating in the class. One of the easiest ways of doing this is by promoting class discussions. At the start of every lecture, there is a 5–7 minute session where students are encouraged to ask questions relating to the course. This provides students with

the opportunity to review previous material and revisit any concerns or questions they may have. This concept can be further extended by having students develop questions based on what they feel is still unclear, and addressing these questions towards the end of the class.

An effective way of promoting an active-learning environment is to have students work in small groups to answer questions, allowing them 2–3 minutes for discussion. The purpose here is to focus their attention on a particular point and give students a chance to process the material by discussing and questioning it. An assortment of real-world examples are used to illustrate programming behaviors. For example to demonstrate the process of developing good algorithm design practices we discuss how the existence of software bugs can contribute to system failures. One case described is the Ariane 5 rocket launched by the European Space Agency in 1996. I first show the class a QuickTime movie of the rocket which disintegrated forty seconds after its lift-off from Kourou, French Guiana. A series of questions are introduced in order to analyze the incident:

- What do you think caused the rocket to disintegrate?
- If I told you the problem was caused by an error in the inertial reference system, compensating for a wrong turn that had not taken place, what could cause this?
- A data conversion from 64-bit floating point to 16-bit signed integer value caused a software exception. Why did this lead to a system failure?
- How could the algorithm have been adapted to avoid such a problem?

This ultimately leads into a discussion on programming errors, and how they can be avoided.

4.3. Interactive programming and algorithm tracing

One of the benefits of using active learning in the delivery of introductory programming courses is the ability to perform interactive programming exercises. Traditionally, programming examples are shown on an overhead slide, and the program is worked through line-by-line in a static manner. Using interactive programming, programs can be written, compiled and run in-situ, allowing the complete process of design and implementation to be illustrated. This gives students the opportunity of experiencing each of the processes involved in implementing a program, including any errors which may occur in the program, and methodologies for tracing these bugs and deriving solutions. Programming logic errors such as infinite loops, or run-time errors such as divide-by-zero can be effectively

demonstrated in this manner. Permitting the student to "create" input allows for experimentation of the algorithm in different scenarios. Most importantly it allows the instructor to answer questions by showing the answer [18].

4.4. Case studies and problem sets

A case study is a story or narrative of a real life situation that sets up a problem for the students to analyze and resolve. A case study must not only introduce new programming concepts, but provides students with new insight into a previously unexplored problem area. The class is given a problem statement and contribute to the process of deriving a solution, first by exacting an algorithm, and then implementing the algorithm. Solutions to the case studies and problems are provided electronically after the class. Students are encouraged to work in small groups and are given 3–4 minutes to formulate an algorithm. The entire class then proceeds to work through an implementation, discussing different design ideas with groups providing rationale for their ideas. A case study involves using knowledge acquired from previous lectures as the foundation to design an algorithm. For example, a case study on the Fibonacci series is used to illustrate the effectiveness of looping structures, and follows the following format:

- A brief history of "Leonardo of Pisa" and his accomplishments
- A background to the conception of the Fibonacci series
- Review of applications of Fibonacci numbers in nature:
  - Pine cones, flowers and bees.
- Description of a method of calculating the Fibonacci series
- Deriving an algorithm
- A review of alternate solutions.

Later in the course, when we have studied recursion, or arrays in the context of C, the students return to this case study and re-implement it. Groups may work on algorithms incorporating different types of loops simultaneously, leading to a discussion on the benefits and shortcomings of each particular approach. Case studies also serve to illustrate how programming paradigms can be used to solve real-world problems. Here we use simple models such as the "Sieve Of Eratosthenes" [19] which looks at identifying prime numbers, to more complex ecological models used to simulate population growth [20]. Each week the preceding week's laboratory or assignment exercise is discussed in class. The nature of each problem is first briefly discussed and then students are invited to propose

solutions. Choosing a particular solution, a discussion ensues which follows a model adapted from Berglund et al. [11]:

- Is the proposed solution acceptable? Does everybody understand the underlying programming constructs used?
- Is the program correct, both with respect to style and output?
- What changes could be made to improve the program?
- Are there any alternative approaches to this problem?

For example, one of our early assignments involves writing a program that performs carbon dating. The program should prompt for the percentage of Carbon 14 remaining in a sample, calculate the age of the sample and print out the result with proper units. The code below is a sample solution:

```
#include <stdio.h>
#include <math.h>

#define lambda 0.00012097

int main( void )
{
    double percent, ratio, age;

    // Prompt the user for the percentage of C-14 remaining
    printf( "Enter the percentage of Carbon 14 remaining: " );
    scanf( "%lf", &percent );

    // Perform calculations
    ratio = percent / 100.0;       // Convert the fractional ratio
    age = ( -1.0 / lambda ) * log10( ratio );  // Get age in years

    // Output the results
    printf( "The age of the sample is: %.2lf years", age );
    return 0;
}
```

We later analyze this program in class, looking at ways that improvements could be made. One such improvement is the addition of code to validate the user input. For example, the percentage of Carbon 14 remaining must be between 1 and 100 percent. A value of 0 would cause the expression `log10(ratio)` to fail, and a negative percentage is inappropriate. This leads to a discussion on which control structure would be most appropriate for performing this validation, and

whether or not the user should be allowed to re-enter the percentage. The code below is an example of the type of construct which could be used:

```
do {
    printf( "Enter the percentage of Carbon 14 remaining: " );
    scanf( "%lf", &percent );
} while ( percent <= 0 );
```

4.5. "5-minute" programs

5-minute programs are similar in concept to minute papers, and are brief, informal, activities geared around designing and/or implementing algorithms related to a distinct problem. They provide students with the opportunity to synthesize their knowledge and evaluate a problem. These exercises help students to reflect upon what they're learning and can facilitate large group discussions. Minute papers allow you to gauge students learning, and they provide a ground for discussion in the next class session. They can be used in combination with the "*muddiest point*" [22] allowing the student to write down the most confusing or ambiguous concepts from the lecture. We often pose them in the form of a "fill in the blanks"-type exercise whereby the students are given a program with key elements of the algorithm omitted. They are then given 5–7 minutes to compose a solution. They can be used at beginning, middle, end of class:

- Beginning: use to probe for difficulties students had with an assignment, to check if they can identify the main points of the previous lecture, or to identify items for discussion
- Middle: use as a "break" in the middle of a lecture to give students a chance for a break, use to see if students can solve a problem using the topic of discussion for that lecture.
- End: use for review, reality check, application, higher order thinking

Consider the following sample "5-minute" programming exercise involving the role of parameter passing in designing functions. The students are given the following program:

```
#include <stdio.h>

#define R 8314

// Calculate the pressure of a tank of CO gas using the
// Ideal gas equation Pv = RT
```

```
void main( void )
{
    double temp;  // temperature of CO gas
    double mass;  // mass of CO gas, in kg
    double vol;   // tank volume in cubic metres
    double vmol;  // molar specific volume;
    double pres;  // pressure

    printf( "Input the temperature of CO gas (deg K): " );
    scanf( "%lf", &temp );
    printf( "The mass of the gas (kg) is: " );
    scanf( "%lf", &mass );
    printf( "The tank volume (cubic m) is: " );
    scanf( "%lf", &vol );

    vmol = 28.011 * vol / mass;

    pres = ideal( vmol, temp );

    printf( "The ideal gas at %.3lf K has pressure ", temp );
    printf( "%.3lf kPa\n", pres );
}
```

They are then asked to design a function `ideal` to calculate the ideal gas equation using pass-by-value. After 5–10 minutes, I ask one of the students to "submit" their solution, and the class as a whole works through the solution to check its validity. The code below is a sample solution.

```
double ideal( double v, double t )
{
    double p;
    p = R * temp / v;   // pressure in Pascals
    return p / 1000.0;  // pressure in kilo Pascals (kPa)
}
```

A subsequent lecture may involve rewriting the function to return the calculated value using pass-by-reference. The code below is a sample solution.

```
double ideal( double v, double t, double &p )
{
    *p = R * temp / v;  // pressure in Pascals
    *p = *p / 1000.0;   // pressure in kilo Pascals (kPa)
}

ideal( vmol, temp, &pres );
```

This activity provides an ideal mechanism for illustrating the notion of program "evolution". You can start the semester designing a algorithm, and using a program skeleton progressively add functionality as new programming concepts are introduced. Case studies and "5-minute" programming exercises are available online at `http://www.uoguelph.ca/~mwirth`.

5. Discussion and conclusion

Lecturing is in itself not the most appropriate way of teaching introductory programming. The rationale for employing active learning in computer science is to provide students with an interactive learning experience, transforming lectures from the usual instructor-oriented model to a more innovative student-centred model. It is based on the key notion of making students active participants in the learning process, rather than passive recipients as might be the case when they are exposed to more traditional lecture-oriented learning [23]. Active learning augments the learning process by influencing the way in which students perceive knowledge. It allows for a departure from the traditional methods of teaching programming which tend to focus solely on teaching programming language syntax.

One of the challenges of incorporating active learning is trying to persuade students to become involved in a large classroom, an environment which can be intimidating for many students. It is important to encourage participation from the onset of the first class, and positive reinforcement over time will increase the students willingness and desire to participate in the process. One concern with active learning is that less material is covered during a semester. Since active learning involves setting aside a portion of class time for learning activities, the amount of material covered in the class will be reduced, but students obtain a deeper understanding of programming paradigms and their application to problem solving. But it may be more appropriate to ask more pertinent questions: "How much material do students retain using the current method?", or "What type of learning occurs within traditional lectures?". Some of these questions focus on digesting knowledge. Traditional lecturing can create a level of understanding that is superficial since the demands of note-taking may preclude the opportunity to analyze knowledge at a deeper level. But does it really work? In the five semesters we have been using this technique in the classroom, we have nearly always seen higher class averages than corresponding semesters taught using more

traditional techniques. The averages for nine consecutive semesters are given below (semesters incorporating active learning are shown in *italics*):

W00	F00	W01	*F01*	W02	*F02*	*W03*	*F03*	*W04*
73	67	65	*78*	70	*72*	*77*	*80*	*82*

Active learning involves various departures from the normal expectations of the passive classroom. Instructor's must learn to evolve from "The Sage" who walks into class, proceeds to the front of the room and dispenses information to "The Builder" who focuses on creating a more learner-centred classroom [24]. Although not every instructor is as receptive to such methods, allowing students to think and voice their ideas is preferential to the rote-learning and expunge which often occurs. Active learning helps students comprehend concepts that are taught, makes a course more interesting and relevant to the students and shows students how to use programming as a tool to solve a wide variety of problems from many varying disciplines. Although oriented towards computer science, many of the techniques discussed can be adapted to other disciplines.

References

[1] J. D. Wilson, N. Hoskin and J. T. Nosek, The benefits of collaboration for student programmers, *ACM SIGSE Bulletin* **25** (1993), 160–164.

[2] T. Jenkins and W. Towle, *Teaching programming to novices – Can technology help?*, in Proceedings of 5th Annual Conference on the Teaching of Computing, (G. Daughton, and P. Magee, eds.), Dublin City University, 1997, 102–104.

[3] D. Laurillard, *Rethinking University Teaching: A Conversational Framework for the Effective Use of Learning Technologies*, Routledge Falmer, London, 2001.

[4] F. Marton, G. Dall'Alba and E. Beaty, Conceptions of learning, *International Journal of Educational Research* **19** (1993), 277–330.

[5] J. Thomas, The variation of memory with time for information appearing during a lecture, *Studies in Adult Education* **4** (1972), 57–62.

[6] D. Bligh, What's the Use of Lectures?, *Intellect*, Exeter, England, 1998.

[7] J. Hartley and I. K. Davies, Note-taking: A critical review, *Programmed Learning and Educational Technology* **15** (1978), 207–224.

[8] V. Magnesen, A review of the finding from learning and memory retention studies, *Innovation Abstracts*, National Institute for Staff and Organizational Development, 1983.

[9] D. Bligh, *What's the Use of Lectures?*, Harmondsworth: Penguin Books, 1971.

[10] B. S. Bloom, *Taxonomy of educational objectives: The Classification of Educational Goals: Handbook I*, Cognitive Domain. Longmans, New York, 1956.

[11] S. M. Taylor, *Cooperative Learning in Distance Education*, Indiana Higher Education Telecommunication System, 1997, http://www.ihets.org/learntech/distance_ed/fdpapers/1997/taylor.html.

[12] L. W. Sherman, *Cooperative learning in post-secondary education: Implications from social psychology for active learning experiences*, in American Educational Research Association, Chicago, 1990.

[13] J. I. Shenker, S. A. Goss and D. A. Bernstein, *Implementing Active Learning in the Classroom. Instructor's Resource Manual for Psychology*, Houghton-Mifflin, 1996.

[14] M. A. Wirth, E-notes: Using electronic lecture notes to support active learning in computer science, *ACM SIGCSE Bulletin* **35** (2003), 57–60.

[15] D. Cordes and A. Parrish, *Active learning in computer science: Impacting student behavior*, in ASEE/IEEE Frontiers in Education, IEEE, Boston, 2002, T2A1-5.

[16] C. C. Bonwell and J. A. Eison, *Active learning: Creating excitement in the classroom*, ERIC Digest, 1991, http://ericae.net/edo/ED340272.htm.

[17] R. E. Sabin and E. P. Sabin, Collaborative learning in an introductory computer science course, *ACM SIGSE Bulletin* **26** (1994), 304–308.

[18] S. H. Rodger, An interactive lecture approach to teaching computer science, *ACM SIGCSE Bulletin* **27** (1995), 278–282.

[19] C. Bayes and R. Hudson, The segmented Sieve of Eratosthenes and primes in arithmetic progression, *BIT* **17** (1977), 121–127.

[20] W. Wilson and W. G. Wilson, *Simulating Ecological and Evolutionary Systems in C*, Cambridge University Press, Cambridge, 2000.

[21] A. Berglund, M. Daniels, K. Lundqvist and E. Westlund, *Encouraging active participation in programming classes*, in Proceedings of 7th National Conference on College Teaching and Learning, 1996.

[22] F. Mosteller, The 'Muddiest Point in the Lecture' as a Feedback Device, *On Teaching and Learning: The Journal of the Harvard-Danforth Center* **3** (1989), 10–21.

[23] T. Jenkins, *A participative approach to teaching programming*, in Proceedings of ITiCSE, ACM Press, 1998, 125–129.

[24] Professors evolving?, *The Teaching Professor* **15** (2001), 7–8.

MICHAEL WIRTH
DEPARTMENT OF COMPUTING & INFORMATION SCIENCE
UNIVERSITY OF GUELPH
GUELPH, ONTARIO N1G 2W1
CANADA

E-mail: mwirth@uoguelph.ca

(Received July, 2004)