A Course Dedicated to Developing Algorithmic Problem-Solving Skills – Design and Experiment

Orna Muller

Software Engineering Department
Ort Braude College of Engineering,
Karmiel, Israel
ornamu@braude.ac.il

Bruria Haberman

Department of Computer Science
Holon Institute of Technology, Holon, Israel;
Davidson Institute of Science Education
Weizmann Institute of Science, Rehovot, Israel
Bruria.haberman@weizmann.ac.il

Keywords: Algorithmic problem-solving skills, algorithmic patterns, analogical reasoning, reflection, instructional design.

Abstract

Undergraduate students often start their academic course of studies with inadequate learning and thinking skills. Our college has a policy of setting high standards and demands, while supporting students' learning in a variety of ways. In this paper we present a distinctive course designed to aid students develop algorithmic problem-solving skills. The course is taught in parallel to a CS1 course and elaborates on activities such as analogical reasoning, prototyping problems, comparison between alternative solutions and reflection on problem-solving processes. It is one of the courses offered at our institution aimed at strengthening general learning and thinking skills, in addition to the regular disciplinary curriculum. Feedback from participants in the course demonstrates a developed awareness and appreciation of abstract ideas beyond programming knowledge. Students report on acquiring problem-solving skills that enable them to cope with compound problems. Additionally, students claimed that they had broadened their repertoire of algorithmic ideas leading to more efficient and elegant solutions.

1. Introduction

The majority of our undergraduate students start their academic course of studies several years after graduating high-school, with unsatisfying learning and thinking skills. The policy of our College is to insist on high level courses and demands, while supporting students' learning in a variety of ways, with the expectation that students with good potential to graduate would cope with the high demands and accomplish high achievements.

The Center for Teaching and *Learning* at Ort Braude College was established five years ago aiming at enhancing students' study skills as well as improving faculty teaching proficiency. The center provides a variety of projects for supporting students such as active learning peer-led study groups, a program for underachievers, a center for active learning, a center for technology support, and others (http://www.braude.ac.il/learn). In addition, courses for developing learning and thinking skills are offered to freshman students.

1.1. Courses for developing learning and thinking skills

The College offers first year students courses for developing learning and thinking skills. Every student is required to choose one of these courses according to her interests and needs. Some of the courses emphasize general learning and thinking skills while others deal with skills related to specific engineering content. The courses were either developed for or adapted to students' needs by the faculty of the college. Some of the courses currently being offered include: Instrumental enrichment (according to Feuerstein’s theory (Feuerstein, Rand, Hoffman & Miller 1980)): developing thinking and learning strategies and creating work habits to improve functioning; Creative mathematical thinking: developing a mathematical “tool box” for solving problems with emphasis on creative scientific thinking; Academic learning skills: coping with increased learning pace and heavy load, stress before exams, developing self-discipline and efficient learning; Learning how to learn, for students with learning disabilities; Systematic inventive thinking: acquaintance with tools to improve analytical processes by raising new ideas in different domains; Development of algorithmic problem-solving skills, which is introduced here.

One main goal of these courses is to enhance students’ ability to cope with engineering studies and thus reduce the drop-out rate. Introducing the learning and thinking skills courses as part of engineering studies in the college constitutes a change in the perception of the role of college as an institution that prepares its graduates for a rapidly changing and developing reality. Specifically, this means improving students' ability to integrate into the workforce, to face future challenges in their professional life, and to become efficient and independent learners.

2. The DAPSS Course

2.1. Motivation and rational

In their first year in college students take courses in Mathematics, Physics and basic disciplinary courses. The Introduction to Computer Science (*CS1*) course is compulsory for freshmen in the Software Engineering and the Industrial Engineering & Management departments. The *CS1* course is multi-purpose: it introduces a new way of thinking (i.e. algorithmic thinking), a programming language (C), and introductory topics related to programming and CS principles (e.g., modularity, complexity of algorithms).

Students frequently have major difficulties in managing the *CS1* course. The failure rate is high and many students need to retake the course. Only about half of the students have some previous experience in programming, usually from their high-school studies. These students manage better with the course than the others, though a significant percentage of them experience difficulties too. Developing solutions to algorithmic problems constitutes a major challenge for novices taking a *CS1* course. Students have difficulties in formulating an idea for a solution, recognizing similarities among problems, and identifying familiar subtasks in a compound problem; consequently, they frequently end up with incorrect and cumbersome algorithms (Deek & McHugh 2000; Du Boulay 1986; Joni & Soloway 1986; Robins, Rountree & Rountree 2003).

We assume that one main difficulty arises from the intensive and rapid progress when studying this loaded course. Although it is the first course in computer science that students take, with rather basic content, they need to learn and gain many different kinds of knowledge and skills. Essentially, novices are expected to become good programmers with fairly high level problem-solving abilities within a period of three months. Unfortunately, students are often overwhelmed by the many new ideas and details they need to comprehend in this short period of time.

There is an implicit perception that algorithmic problem-solving skills are acquired while learning to program. Programming assignments in the *CS1* course exercise the use of programming structures, but they also require the development of algorithms. Experience shows that while students handle well loops and arrays, they often demonstrate poor problem-solving skills; difficulties arise with regard to problem decomposition, developing an efficient solution, or using previously seen solutions even for elementary problems (Ebrahimi 1994; Joni & Soloway 1986; Perkins & Martin 1986; Spohrer & Soloway 1989).

One of the main obstacles in learning problem solving while learning programming is the tendency to put a lot of attention on syntax rules while missing the abstract ideas. We believe that targeting learning abstract ideas and problem-solving principles within a separate course (in parallel to learning programming) may enhance the development of students' problem-solving skills, and support students' learning.

These considerations motivated the first author to suggest the *Development of Algorithmic Problem-Solving Skill*s (*DAPSS*) course as part of the courses for improving learning and thinking skills (in addition to the regular curriculum of the engineering department a student belongs to).

The development of the *DAPSS* course was part of an initiative of the national Ministry of Education to develop courses with orientation of enhancing thinking skills intrinsic to some subject matter. Due to the limited transfer of skills made by learners between areas found in several observations and research (Bassok 2003; Mayer & Wittrock 1996; Robins 1996), the emphasis of this initiative is the development of skills in the context of a specific area. Accordingly, the *DAPSS* course is recommended to be taken in parallel to studying *CS1*. The main goal of the course is to put aside the details of the programming language and concentrate on general concepts, ideas and processes. It elaborates on the development of algorithmic problem-solving skills.

Course development was motivated by the positive results of previous research (Muller 2007; Muller, Haberman & Ginat 2007; Muller & Haberman 2008): Algorithmic Patterns were introduced to students in a CS introductory course in which a pedagogical approach called Pattern-Oriented Instruction was utilized. According to the Pattern-Oriented Instruction approach, learning algorithmic patterns may reinforce the creation of cognitive schemas associated with expertise and knowledge organization. Research findings demonstrate that this approach facilitates students' ability to develop correct, efficient, and elegant algorithms, especially among those students with superior mathematical capabilities (Muller 2005). The *DAPSS* course is an attempt to implement these findings in the context of a course aimed at improving thinking and learning skills.

2.2. Instructional Design of the Course

Course planning included a process of defining the skills that the course intended to develop and the instruction methodologies most suitable for supporting the acquisition of those skills.

The course includes a structured introduction to Algorithmic Patterns derived from generic "expert" solutions to recurring algorithmic problems. Course organization is based on solving a well-planned selection of problems. Most importantly, the choice of problems is driven by the creation of opportunities to explicitly demonstrate and practice different skills (see Table 1), particularly aspects of analogical reasoning, problem analyses, abstraction processes and verbalization of ideas. Problems are grouped according to prototypes, such as *Searching for an Item* or *Finding an Extreme Value* in a list. Prototype problems categorized according to various algorithmic patterns. Problems of the same prototype are presented in various contexts and applications. As the course progresses, problems become more challenging and it is usually composed of several subtasks with interrelations between them. The following major types of relations between subtasks are presented and discussed: *serial*, *by inclusion* and *by interweaving*. We consider the identification of a problem's structure an important aspect of abstraction during problem-solving (Muller & Haberman 2008). Examples of prototype problems and the way they are introduced and analyzed according to the POI approach are described in previous papers (Muller, Haberman & Ginat 2007; Muller & Haberman 2008).

Course setting is mostly based on workshop activities, and the group of students is limited to 30 students at most. A large portion of the course is devoted to solving algorithmic problems, analyzing, comparing and discussing students' solutions to problems, while reflecting on thinking and problem-solving processes. In particular, various kinds of class and homework assignments are introduced to students, such as: categorizing problems, composing analogical problems to a given problem, abstracting a prototyped solution out of several analogical problems. Algorithms are written in pseudo-code according to basic conventions which are introduced in the beginning of the course. One of the guiding lines of the *DAPSS* course is to disregard syntax details of a programming language.

We would like to note that efforts to introduce basic algorithmics in a class preceding a *CS1* course (called CS0) are not rare (see for example reports in (Dierbach, Taylor, Zhou & Zimand 2005; Doyle 2005)). Courses of this type mentioned in the literature usually involve introduction to computers, programming and the use of software development environment as well. Contrary to a "typical" *CS0* course, the *DAPSS* course is not part of the regular software engineering curriculum but one of several courses directed towards helping students to improve their general learning and thinking skills. It is taught in parallel to a *CS1* course and its objectives and design go beyond introduction of algorithms. The *DAPSS* course puts special emphasis on reflective processes, awareness to problem-solving behaviour and development of cognitive skills (e.g. aspects of analogical reasoning and analysis of common difficulties).

Table 1 illustrates skills and associated supporting activities according to which the course is organized.

Problem-Solving Skill	Learning/Instructional Activity
Problem's comprehension	Reformulation of the problem statement in terms of initial state, goal, assumptions and constraints.
Problem's decomposition	Identifying, naming and listing subtasks.
Analogical reasoning	Identifying similarities between problems. Distinction between structural and surface similarities. Raising awareness of common mistakes caused by referencing to unsuitable problems.
Generalization and abstraction	Extracting prototypes of problems from analogical problems in different contexts.
Identifying problem's prototype	Systematic and well-structured introduction of algorithmic patterns. Problems' categorization.
Problem's structure identification	Identifying the relation between subtasks. Schematizing a problem's structure, using diagrams.
Evaluation and appreciation of efficiency and elegancy	Comparing solutions with regard to efficiency and elegancy.
Reflection and drawing conclusions	Reflecting on problem-solving processes and strategies. Making conclusions for the future.
Verbalization of ideas	Formulation of a precise idea, differentiating between an idea and its implementation.

Table 1. Problem-solving skills and associative activities

3. Course Evaluation

The *DAPSS* course is currently being taught for the fourth time. Altogether 85 students participated in the previous three courses. Course design is constantly under examination in order to enrich its contents and adopt it to students' needs. At this stage we have been trying to collect and evaluate students' reactions and feedback. The major source of students' input has been a questionnaire that was administered to them at the end of each course. Students were asked to reflect on their experience and give their opinion regarding: (a) the main skills they acquired in the course; (b) the content and structure of the course; and (c) the desirable relationship between this course and the *CS1* course (either to keep the courses separated, to take one before the other, or to unite the courses). All questionnaire items were open questions since we have been mostly interested in students' reflection and expression of thoughts and attitudes.

The analysis of answers given by students required a few attempts to characterize them according to different criteria, until we identified several main categories of repeated ideas and comments. The following findings provide a foundation for further quantitative inquiry planned for the course currently being taught. A new version of questionnaire that quantifies perceptions of the course will be given to the students at the end of the semester.

3.1. Findings

We have observed that students' comments referred to the following issues: (1) Acquisition of problem-solving skills; (2) Awareness of various abstraction levels in problem solving (3) Cultural aspects of problem solving, and (4) Design of instruction perspectives.

Here we exemplify students' views related to these issues:

(1) Acquiring general problem-solving skills

Students expressed confidence that when being introduced to a novel problem they have tools to analyze and solve it: *"The course improved my ability to cope with compound problems"*; *"I've gained skills to recognize in the big problem the small sub-problems we've practiced*".

They mentioned skills related to analogical reasoning: *"The course developed my ability to see similarities between problems*"; *“We have learned to identify a problem's prototype"; "Naming a problem's prototype means that we alreay know something about the idea behind the solution".*

Some of the skills and gains they mention are of a general nature: *"The course sharpened our thinking"; "We have learned to express ideas”.*

Notes made by several students rely on their experiences during the *CS1* course they are taking in parallel: *"Many times I've solved problems (in the CS1) quicker than others, and when I analyzed that, I recalled that we have discussed similar problems during this (DAPSS) course".*

(2) Awareness to distinct abstraction levels

Students' answers reflect on their ability to distinguish between two major levels of problem solving: the abstract level (i.e. problem analysis and solution design) and the concrete level (i.e. writing code and running programs). The distinction between these levels is clearly demonstrated in the following sample of students' statements: "*The course enabled me to see that algorithms are something separate from programming languages"; "It gave me an opportunity to identify problem solving as something apart from syntax of a specific language"; "Each problem actually consists of two tasks: the first is to find the idea for a solution and the algorithm for solving the problem, and the second is a programming task of how to translate the algorithm into code".*

Students pointed to different stages of the problem-solving process. For example: “The course helps in understanding the systematic nature of a problem-solving process”; "After identifying the problem's starting point, goal, idea for a solution and the type of a problem, the problem becomes much simpler".

Students developed awareness of the importance of learning theoretical and abstract aspects of problem solving: "The ability to develop an algorithm without being tied to a certain programming language helps solving problems in any language"; "It is important to put emphasis on the more theoretical sides of programming, for example, how to approach a problem, how to categorize problems, etc., all the rest is technique"; "People with no background need this (DAPSS) course more than working on techniques”.

(3) Cultural aspects of problem solving

Students expressed ideas that demonstrate a shift towards an "expert” view, such as realizing the importance of an appropriate approach to problem solving, problem comprehension, and analysis and "thinking and planning before implementing". They dropped the typical view of beginners that "what runs well is good enough".

Students reflected on a shift towards mature norms of approaching a problem. For example: "My thinking and the way I approach a problem became more ordered"; "The course taught me that when approaching a problem I first think of an idea for a solution and don’t immediately start writing code";

"When starting to write code before analyzing the problem well and without having a good idea, the solution, if finding any at all, will be complicated and not elegant".

Almost all students mentioned the contribution of the course to broadening their repertoire of algorithmic ideas: "Although I had previous experience in programming, this course contributed significantly to learning ideas for solutions. I have learned new ideas I wouldn't get to alone, such as the use of "bubbling" to move the largest element to the end of a sequence ".

Many students mentioned their appreciation of ideas that lead to clear, straightforward, elegant and efficient algorithm. A sample of their answers regarding this issue: "The biggest challenge for me is to find an idea which is hard to think of"; "To know how to develop a good, elegant, short, clear and smart algorithm is the biggest hurdle, and it is something that we need to work on and develop from the beginning…"; "It is not hard at all to solve problems in long and awkward ways. The challenge is to find a short and efficient solution".

(4) Design of instruction perspectives

Students were relating to two main aspects of course design: (a) the course content and organization, and (b) the recommended relationship between the DAPSS and the CS1 courses.

Students stated that the course contributed to the construction of their knowledge in a meaningful way: "The organization of knowledge helps in categorizing a problem according to its prototype".

One aspect of course design that students mostly benefited from is grouping problems according to their prototypes: "The way the problems were organized in this course has a definite influence on learning problem-solving…"; "Dividing the course into themes and seeing examples for each type of problem helps in pulling out from the toolbox the right way to solve a new problem"; "It is important to exercise the same type of problem several times, so a student better understands the main common idea in those problems. Otherwise, we cannot make links between them and cannot assimilate what is taught"; "The more we see examples and ideas, the more we are able to connect between similar problems; it is an experience we should gain from the beginning"; "When solving 'the same problem' several times, each time phrased in a different manner, it develops the skill of comparison between problems".

Another aspect of course design that students benefited from is the gradual increase in problem complexity which may enhance the skill of problem structure identification: "Learning types of small problems helps in visualizing a big problem"; "Often in the course, problems contained smaller problems similar to those we had learned previously… This method showed how important it is to decompose a task into subtasks, and to handle each subtask separately."

Students compared the different emphasis of the two courses (programming structures versus algorithmic ideas): "Letting students become acquainted with programming structures and presenting them with problems without being familiar with different types of problems is not enough in order to give students the tools to cope with problems"; "I think that arranging course topics by ideas and not by programming structures is very significant since it is the basic theory which gives the tools for understanding. Learning structures such as loops is only a tool for implementing ideas"; "In a CS1 course there is a limited emphasis on algorithms and efficient solutions to problems"; "…This (DAPSS) course put more emphasis on the efficiency of algorithms".

All students who answered the reflective questionnaire (N=45) considered the course important and recommended that it should keep being taught.

The students (all but one) described their views regarding the preferable relationship between the two courses; 49% thought that the contents of the DAPSS course need to be integrated into the CS1 syllabus. Other 49% thought that it was right to keep the courses apart. Students' main arguments for separating the two courses were: "The two courses need to be kept separate, since the CS1 course is loaded and intense, and it seems impossible to add new topics to it. In addition, if this (DAPSS) course was part of the CS1 course, we will still be busier with implementation in C and not with problem-solving issues".

Sixty seven percent of the students in favour of separating the courses recommended that they will be taken at the same semester: "The two courses need to be given in parallel. This (DAPSS) course is more theoretical than practical – it works more on ideas and methods for solving problems, and exposes us to different types of problems, while in the CS1 course these ideas are executed". The rest (33% of that group) recommended that the DAPSS course should precede the CS1 course since it could be regarded as a preparatory course.

4. Discussion and Concluding Remarks

CS1 courses are considered as an introduction for newcomers to the field. These courses aim at presenting fundamentals of computer science. One particular goal is to initiate the development of algorithmic problem-solving skills and professional habits and norms. Since further CS courses are built upon the fundamentals of a *CS1* course, it is of great importance that the topics studied be well-understood and assimilated by the students. In particular, principles of algorithmic problem solving should be emphasized. However, this goal is not easy to achieve while studying *CS1*: learning a programming language implies spending much time on syntax and technical aspects of programming at the expense of concentrating on abstract ideas and principles (Du Boulay 1986).

The *DAPSS* course is unique among other courses aimed at "developing thinking and learning skills" which freshmen are required to enrol in at the Ort Braude College of Engineering. Such courses, being an integral part of the curriculum, reflect the importance that the college attributes to cultivating general thinking skills and problem-solving abilities among students.

An important question is whether it is suitable and practical to raise, in this early stage of students' studies, aspects of the quality of algorithmic solutions alongside correctness (i.e. efficiency, simplicity and elegancy) (Ebrahimi 1994; Joni & Soloway 1986). According to our teaching and research experience, proper guidance towards developing simple and elegant solutions, may assist novices in avoiding cumbersome solutions. Moreover, focusing on these aspects may trigger students' motivation and interest in algorithmic problem solving.

The *DAPSS* course enables concentrating on algorithmic problem solving, unrelated to specific programming languages. The course is devoted to the development of thinking and learning skills in the context of algorithmics; hence various aspects, such as abstraction and generalization, prototyping, evaluation of solutions, and aspiration towards simplicity and elegance, are conceptualized, discussed and practiced. Course put emphasis on cognitive aspects in an explicit manner. In addition, since the development of problem-solving skills is treated in the context of solving algorithmic problems, and not in some general context as frequently done, some skills (mostly related to an increased awareness to ideas and processes) may be better transferred to the *CS1* and other related courses.

The reflective questionnaire presented to the students at the end of the course allowed us to learn about students' attitudes regarding the influence of the course on their problem-solving skills, and their perception of problem solving in general. The importance of such a questionnaire is two-folded: in addition to gathering study data, it triggers students' retrospective evaluation of the course and its outcomes in terms of personal gains.

Students expressed their feeling that they have improved their problem-solving skills, acquired a repertoire of algorithmic ideas and increased their confidence in approaching a problem. Specifically, students demonstrated awareness to various stages and levels of abstraction in problem solving, and showed ability to abstract general ideas and principles out of the *"ocean of details"* (as expressed by one student). Students appreciated the importance of acquiring general problem-solving skills, and gave legitimacy to learning such skills, independently and beyond learning programming.

Students also acquired common terminology related to problem solving that enables them to express their thoughts, and also, to analyze and point out their difficulties and missing skills; this is a significant step towards improving and gradually gaining proficiency.

One of the goals of this inquiry was to find out whether separating the issue of developing algorithmic problem-solving skills from learning to program has itself any impact. About half of the students recommended that the *DAPSS* and *CS1* courses will be kept separated since each of the courses emphasizes different aspects. The majority of these students suggested that the courses should be studied in parallel; minority of them thought the *DAPSS* should precede the *CS1* course. According to the two views, students may use ideas, tools and skills they gained in the *DAPSS* course when solving problems in the *CS1* course. From our pedagogical perspective, another reason for taking the courses in parallel is the importance for students with no previous programming background to be running programs, so the notion of algorithm and concepts such as variables and loops will be less abstract and better understood.

At this stage, student feedback gives us reason to believe that the course has a positive effect. Nevertheless, we would like to further evaluate the influence of the *DAPSS* course on students' ability to solve problems introduced in a *CS1* course. We currently are designing the second stage of our study aimed at comparing students who took the *DAPSS* course to students who did not take the course with regard to attitudes, behaviour and success in algorithmic problem solving. We plan to use a methodology of structured personal inspection and interviews as a means of evaluation. Since students enrolled in the course are varied with regard to their preliminary programming knowledge and experience (e.g. part of them had taken a CS course in high-school), we intend to compare sub-groups of students with similar programming backgrounds and personal abilities. A comparison of several homogeneous sub-groups may illuminate on possible differences in the impact of this course on students with different profiles.

5. Acknowledgments

Development of the course was supported by the Pedagogical Council of the Israeli Ministry of Education.

6. References

Bassok, M. (2003). Analogical transfer in problem solving. In Davidson, J.E. and Sternberg, R.J. (Eds.), The psychology of problem solving. Cambridge University Press.

Dierbach, C., Taylor, B., Zhou, H., & Zimand, I. (2005). Experience with a CS0 course targeted for CS1 success. Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, ACM Press, 317-320.

Deek, F.P., & McHugh, J. (2000). Problem-solving methodologies and the development of critical thinking skills. Journal of Computer Science Education, 14(1-2), 6-12.

Doyle, J.K. (2005). Improving performance and retention in CS1. Consortium for Computing Sciences in Colleges CCSC: Midwestern Conference, JCSC 21(1), 11-18.

Du Boulay, B. (1986). Some Difficulties of Learning to Program. J. of Educational Computing Research, 2(1), 57-73.

Ebrahimi, A. (1994). Novice programmer errors: language constructs and plan composition. International Journal of human-Computer Studies, 41, 457-480.

Feuerstein, R., Rand, Y., Hoffman, M., & Miller, R. (1980). Instrumental Enrichment. Baltimore, MD: University Park Press.

Joni, S.A. & Soloway, E. (1986). But my program run! Discourse rules for novice programmers, J. Educational Computing Research, Vol. 2(1), 95-126.

Mayer, R.E., & Wittrock, M.C. (1996). Problem-solving transfer. In D.C. Berliner & R.C. Calfee (Eds.), Handbook of educational psychology. New York: Macmillan.

Muller, O. (2005). "Pattern Oriented Instruction and the Enhancement of Analogical Reasoning". Proceedings of the 1st International Computing Education Research Workshop (ICER), Seattle, Washington, 57-67.

Muller, O. (2007). The effect of pattern-oriented instruction in computer-science on algorithmic problem-solving skills. Unpublished doctoral dissertation, Tel Aviv University, Israel. (in Hebrew)

Muller, O., Haberman, B.& Ginat., D. (2007). Pattern-oriented instruction and its influence on problem decomposition and solution construction. The 12th Annual Conf. on Innovation and Technology in Computer Science - ITiCSE'07, Dundee, Scotland, 25-27, June 2007, 151-155.

Muller, O. & Haberman, B. (2008). Supporting abstraction processes in problem solving through pattern-oriented-instruction. Computer Science Education, 18(3), 187-212.

Perkins, D.N. & Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers, In Empirical Studies of Programmers, E., Soloway, Y., Iyengar (eds.), Albex Publishing Corporation, Norwood, New Jersey, 213-229.

Robins, A. (1996). Transfer in cognition. Connection Science, 8(2), 185-203.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: a review and discussion. Computer Science Education, 13(2), 137-172.

Spohrer, J.C., & Soloway, E. (1989). Novice mistakes: are the folk wisdoms correct? In E. Soloway & J.C. Spohrer (eds.), Studying the novice programmer, 410-416. Hillsdale, N.J: Laurence Erlbaum.