Teaching Formal Methods in the Context of Software Engineering

Shaoying Liu
Department of Computer Science
Faculty of Computer and Information Sciences
Hosei University
3-7-2 Kajino-cho Koganei-shi
Tokyo, 184-8584, Japan
sliu@hosei.ac.jp

Kazuhiro Takahashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
ktaka@signal.co.jp

Toshinori Hayashi
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
hayashit@signal.co.jp

Toshihiro Nakayama
Research Center
The Nippon Signal Co., Ltd.
1836-1 Oaza Ezura, Kuki,
Saitama 346-8524, Japan
nkym-ts@signal.co.jp

Abstract: Formal methods were developed to provide systematic and rigorous techniques for software development, and they must be taught in the context of software engineering. In this paper, we discuss the importance of such a teaching paradigm and describe several specific techniques for teaching formal methods. These techniques have been tested over the last fifteen years in our formal methods education programs for undergraduate and graduate students at universities as well as practitioners at companies. We also present a curriculum to systematically introduce formal methods to students at university and a successful program of teaching formal methods to industry. Our experience shows that students can gain confidence in formal methods only when they learn their clear benefits in the context of software engineering.

Categories and Subject Descriptors: K.3.2 [Computers and education]: Computer and Information Science Education – Computer science education, Curriculum. D 2.4 [Software engineering]: Software/Program Verification – Model checking.
General Terms: Reliability, Standardization, Verification.
Keywords: Formal Methods, Education, Software Engineering, Teaching Methods, Formal engineering methods

1. INTRODUCTION

Despite more than forty years of effort to develop various theories, languages, methods, and tool supports, practical software engineering is still like a "desert", lacking directions and effective ways of finding the way out of the software crisis. Formal methods were developed to address this problem by providing mathematically-based techniques, including formal specification, refinement, and verification. In theory, we now know how to use formal notations to write specifications, use refinement calculus to gradually transform a specification into a correct implementation, and use Hoare or Dijkstra's logics to prove programs correct with the same degree of the rigor that we apply to mathematical theorems. However, none of these techniques is easy to use by ordinary practitioners to deal with real software projects. The problem is the complexity and the incapability of formal methods in dealing with large-scale systems and frequent changes in requirements and designs in practice.

Having said the above challenges in directly applying formal methods, we do not mean that formal methods are useless. In fact, they are more necessary than ever when more and more software systems are embedded into systems deployed in many places of our society, but their role is different from other software techniques. The role of formal methods is education, and their power can be transferred to software engineering projects through the developers who have learned and mastered them. The way to use formal methods in practice is formal engineering methods [1], not formal methods. For example, the SOFL formal engineering method provides a three-step approach to constructing formal specifications to help requirements analysis and system design, and specification-based inspection and testing for detecting bugs in both specifications and programs [2]. Software projects are human activities; they must be completed by required time and within specified budget, and they often face the instability of development teams. In such a situation,

completely applying formal methods is rarely practical, but the improvement of software quality can be realized by equipping the developers with a disciplined manner and rigorous way of thinking through formal methods education.

To encourage more students, both inexperienced and experienced in software development, to learn formal methods, we must first build up their motivation by demonstrating the clear benefits of formal methods in improving current software engineering practice. While this is rather challenging due to the lack of reliable quantitative evidence in industry, many empirical studies, some of which were done in industrial setting [3,4,5], may be used for this purpose. To let students enjoy learning formal methods, excellent teaching styles and techniques, sensible curriculum arrangements, and academia-industry collaboration will be the key of success. In this paper, we describe several techniques for teaching students formal methods, an effective university curriculum, and a successful program for teaching formal methods to industry. Our fundamental idea is to put the formal methods education in the context of software engineering, because our interest is in the issue of how to foster software engineers for industry. Of course, as Parnas pointed out [6], formal methods should not be restricted to software engineering, but linked to and integrated in general engineering mathematics. Since the issue of general application of formal methods is beyond the scope of this paper, we focus our discussions on the issue of teaching formal methods for software engineering.

The remainder of this paper is organized as follows. Section 2 concentrates on discussions of teaching techniques, the most important factor of the three in the key of success mentioned above. Section 3 describes the current curriculum arrangement in the Department of Computer Science at Hosei University, which proves to be effective. Section 4 presents a successful program for teaching formal methods to industry. Section 5 discusses the importance of continuing education. Finally, in Section 6 we conclude the paper.

2. TEACHING TECHNIQUES

In this section, we introduce some specific techniques for teaching formal methods. These techniques have been tested by the first author over the last fifteen years of teaching VDM [7], SOFL [1], and Morgan's refinement calculus [8] at universities and companies.

2.1 Starting with Examples

Learning formal methods is similar to learning other theories or techniques, students like to start with simple examples. These examples must come from the daily life and must be able to link the problem in practice to a potential formal methods solution. This way of teaching will motivate students and build up their interests in formal methods. For example, when explaining the ambiguity problem in informal specifications and the fact that it can be resolved by formalization, we often use an operation for searching for an integer in an integer list as an example. After explaining the impreciseness of the informal requirement statements, we present a formal specification which is both precise and concise. This example helps students understand the potential power of formalization.

2.2 Gradual Introduction to Important Concepts

The fundamental concepts are the key to understand the spirit of formal methods. It is quite effective to help students understand the essential principle of formal methods if sufficient efforts are made to teach the concepts. For example, when introducing formal specifications, we focus on the illustration of pre- and post-conditions. An effective way to teach the pre-post concept is by comparing them with the corresponding algorithm and let students understand the real difference and relation between a specification and an algorithm. The comparison can be made on the basis of simple scientific computation. For example, we often use the operation for yielding the square root of an integer as an example. The pre-condition of the operation is $x \geq 0$ and the post-condition of the operation can be $y^2 = x$, where x is input and y is output. But the corresponding algorithm would be something like y = Math.sqrt(x). This example gives rise to a problem that output y produced by the algorithm may not satisfy the post-condition of the operation because the algorithm obtains only an approximation of the real square root of some positive integers. In this circumstance, it is useful to tell the students the importance of noticing this inconsistency between the specification and the implementation. This is also a good example to show the need for using or building proper theories in the application domain.

2.3 Massive Exercises on Basics

Efficiently writing accurate formal specifications requires the developer to have a good understanding of features of various data types and high skills in applying the well-defined operators on the data types, such as boolean, set, sequence, and map types. Therefore, massive exercises on the basic operators must be done by students. The most effective way to incorporate exercises into the teaching program is to let students do exercises immediately after a data type is introduced. For example, after the introduction of the set types, students must learn the meaning of the operators, such as union, intersection, cardinality, membership, subset, proper subset, and so on by applying them to specific set values. If time allows, a public discussion on students' results is helpful. According to our experience, such a discussion can help capable students find out the reason for their mistakes and ordinary students find out the correct way of thinking. This training is similar to the basic training in sports. To be an excellent football player, for example, one must run fast and have a strong

body. To build up these qualities, he or she must spend much time and make great efforts in the basic training. Anybody who ignores the basic training will fail to perform satisfactorily in matches..

2.4 Teaching Specification Patterns for Abstraction Skills

Effectively using a formal method requires the developer to have high skills and ability in mathematical abstraction, especially in the context of software development. How to help students strengthen their abstraction skills and ability therefore becomes an important issue in formal methods education. While this has been recognized widely as the most difficult thing in teaching, we have gained sufficient knowledge and understanding through our long time teaching experience. Considering the fact that the basic operations required in a software system usually include searching, sorting, merging of two collections of objects, adding some elements to a collection of objects, eliminating some elements from an existing collection of objects, updating some elements from an existing collection of objects, mathematical computation, and their combinations, we put the emphasis on the teaching of how to express all of the above functions using appropriate data types and their related operators. Each of such expressions will form a specification pattern that will remain in students mind and available for application in real software development. For example, what are possible specification patterns for a function which tests that a collection of integers is empty? To answer this question, we first define a collection of integers as a set and a sequence in SOFL (or VDM), respectively, such as intset: set of int and intseq: seq of int. We then discuss the most commonly used specification patterns for each of the data abstractions. For example, for the set of integers, we can use the following patterns to express the fact that the set is empty: intset = {} and card(intset) = 0. Of course, we could have more patterns to express the same meaning, but those would be much more complex and no good for readability. For instance, a possible pattern can be: forall[x: int] | x notin intset. It is up to the teacher to decide whether to discuss such a complicated pattern within the required teaching time. In the case of a sequence of integers, we can use the following patterns to express the fact that the sequence is empty: intset = [] and len(intset) = 0.

After each basic specification pattern is mastered by students, we can then go further to explain how such basic patterns can be applied in a more complicated situation. Let us take an operation to search for an integer in a collection of integers as an example. To explain how such an operation is specified, we take the same approach as the one to teaching the basic patterns by first defining the collection of integers as a set of integers and a sequence of integers, respectively, and then explaining how the operation can be specified by combining the basic patterns for each of the data abstractions.

2.5 Practice through Small Projects

While the basic training is important in teaching and studying formal methods, we should never forget to give students opportunities for linking formal methods to software engineering. In other words, they need to be taught how formal methods will possibly help them in software development practice; otherwise, students (perhaps with some exceptions) will likely to lose the motivation of learning or applying formal methods in practice. The most effective way for this is to let students conduct small projects. For example, after the introduction of VDM-SL and massive exercises on the basics, we can ask students to do one or two small projects. One project can be the construction of a formal specification for a small library system, and another possibility is to let students complete a formal specification for an ATM software. Through such small projects, students can really feel how formal specifications can be built and organized in real software development projects. Of course, such a practice may also give students an opportunity to find the weakness of the specification language they are using. For example, lacking an intuitive mechanism for structuring a whole system in a structured manner in VDM could be found by students. The answer to this problem is to introduce the SOFL specification language to them, since SOFL has solved this problem by using intuitive and formalized data flow diagrams and process decompositions. In fact, many existing formal notations focus only on one aspect of the problem in software engineering and ignore the others, but a real software project needs to take care of all possible aspects. If a method or technique merely helps solve one problem but create more other problems in the context of software engineering, it is unlikely to be popular among practitioners and to be applied in real projects. In this regard, the SOFL method has shown to be the exception, because it provides a systematic and rigorous process to integrating formal techniques into existing software engineering practices and creates no more problems.

2.6 Teaching Formal Methods Using Formal Engineering Methods

The ultimate goal of teaching formal methods (FM) is to create possibility of students applying them in practice. Formal engineering methods (FEM) show how FM can be applied in real projects. One of the very important aspects of FEM is the emphasis of combining diagrams, formal notation, and natural language in a coherent and systematic manner for writing specifications [1]. The purpose of this is to help developers easily understand the specifications they are writing and the specifications written by others. Visualization is intuitive and suitable for describing the overall idea and system architectures; formal notation has a strength to achieve preciseness of statements in specifications; and natural language can be used to provide a friendly interpretation of formal expressions. In general,

FEM differs from FM in that FM tries to answer the question ``what should we do and why'' in software development, but FEM tries to answer the question ``what can we do and how?''. To this end, FEM focuses on techniques and methods for integrating formal methods into the entire process of software development so that the strength of formal methods can be utilized in practice and their weakness of being complex can be avoided. FEM offers how software systems, including all level documents, are actually created and expressed formally, not just a simple mixture of formal notations with pictures. Since a detailed introduction to FEM is beyond the scope of this paper, we refer the reader to the SOFL book [1] for a comprehensive description of FEM.

In fact, the same principle of FEM can also be effectively applied to the teaching of formal methods courses, since teaching is actually a kind of software project whose product is educated students. For example, when explaining a mathematical expression, such as Z = X union Y, we can use a graphical representation (e.g., Venn diagrams) to illustrate the union operation, and at the same time use English, for instance, to explain the meaning of the operation. When introducing an operation in VDM, we can draw a process as we do in the SOFL language to show the input, output, and external variables, but the details of the function of the operation are defined using pre- and post-conditions. With informal explanations in English, the meaning of the whole operation specification can be easily digested by students.

2.7 Tool Support in Education

Almost all of us may have experienced using tools in teaching programming languages, such as Java and C, and found that it is effective to help students write, execute, and test programs (they need many pre-defined packages). Many of formal methods educators apply this idea to the teaching of formal methods courses as well. However, our experience in teaching both VDM and SOFL courses, which focus on formal specification techniques, suggest that using tools in teaching formal methods is not necessarily effective; perhaps less effective than not using tools in some circumstances. There are two reasons. One is that learning formal methods requires students to learn both syntax and semantics of the related specification language. The most effective way for students to remember them is to write formal specifications by hand, as they learn English as a foreign language. It is feasible, because exercises assigned to students in classes are of small scale. It is also effective in strengthening students' memory of the syntax and in deepening their understanding of the abstraction techniques, because students would have no chance to "copy and paste" without thinking by themselves, as we often do on a computer. Another reason is that the purpose of writing a specification is not for a computer to directly run it, but for people to read and understand. Therefore, letting them write a good style of formal specifications by hand is much more helpful for learning than by using a tool to automatically improve the style and format of their specifications. In the case of programming, without a tool, such as a compiler, we cannot run the program. But in the case of writing a specification, there is no need to run it, so without a tool support will not create any significant inconvenience. Instead, for some students who do not want to study formal methods, tool support will create chances for them to "copy and paste" without thinking.

Having said the above, it does not mean that tool support is not necessary for using formal methods in practice. On the contrary, tool support is crucial for improving productivity and reducing chances of creating mistakes in practical developments. For this reason, we let students use a supporting tool, such as IFAD VDMTools or SOFL GUI editor, when they carry out a small project, after a systematic learning of formal specification techniques in classes. This way also has an effect that students feel extremely happy with the tool offering high automation in both writing and analyzing specifications. They have this kind of feeling because they have gone through a hard time in learning formal methods by hand. This is similar to the situation where a person feels happy when he or she has a chance to eat delicious food after a long time starving.

2.8 Dealing with Time Constraint

Mathematical concepts and expressions usually require students to take time to digest, the teaching of them should take slow pace with many examples. However, a course is like a software project: it also has time constraint. As a teacher, we often face a dilemma. On the one hand, we want to teach more contents which are all important for studying formal methods, but on the other hand, we do not have enough time. To tackle this problem, our experience suggests that each course should not be too ambitious; instead, it should be focused. For example, we can teach formal specification, refinement, and formal verification in three different courses, and it would be effective to focus the teaching in each of them on the most fundamental but important parts and give students sufficient time for them to apply the learned techniques. For example, when teaching SOFL, particularly techniques for writing formal specifications using pre- and post-conditions, to students, we usually take the interleaving approach: teaching concepts and asking students to practice using them. After finishing the whole course, we ask students to carry out a small project in which all knowledge learned is required to use. Such a way provides students with many opportunities to learn how theoretical results can be effectively applied in practice.

3. A SYSTEMATIC CURRICULUM

While teaching techniques are crucial to a successful education in formal methods, a sensible arrangement of curriculum concerning formal methods education also plays an important role from an overall view of education. We

have adopted a systematic, step-by-step curriculum for formal methods education in the Department of Computer Science at Hosei University. Although it is still not perfect yet in the sense of involving all students in the department, the curriculum has proved to be effective and successful among the students who have received the education according to our experience over the last nine years of practice.

The curriculum is arranged as follows. We introduce simple VDM-like operation specification techniques to the first year students of four-year undergraduate education, more complicated specification techniques to the second year students, and a systematic formal engineering methods course to the third year students, and finally require the students in the Software Engineering Laboratory for Dependable Systems (SELDS) to apply the learned formal techniques in their graduation projects. All of these classes are optional. For the students who proceed to the postgraduate program, we teach them formal refinement or verification in one course, depending on the background of students in the class (it is possible to have students in the class who did not choose the above formal methods classes during their undergraduate study).

We start introducing the knowledge of SOFL process specification in pre- and post-conditions in the second semester (one year has two semesters: Spring and Autumn semesters) to first-year students. In the first semester, students are required to study programming language Java. Although Java is an object-oriented programming language, we focus on the teaching of the basic control structures (e.g., assignment, sequence, selection, and iteration) rather than on object-oriented features (e.g., classes, inheritance). Students are given two to three programming problems a week and the functional requirement of each problem is written in Japanese (non-structured Japanese). This study gives students a chance to build up basic perception of programming and computation, which will facilitate us to introduce simple specification techniques in the second semester. When teaching the concept of process specification in pre- and post-conditions, we take a step-by-step approach to start with simple specifications and then gradually increase the degree of complication. After a short explanation, students are required to first digest the specification and then implement it in Java, and finally to test it on the basis of the specification. Surprisingly, we found that most of the students in the class can quickly understand the concept of specification and implement it without problems.

In the second year, we introduce more complicated expressions in the SOFL process specification language, such as let-expression, if-then-else expression, and case-expression, and the composite and set types to improve expressive power of formal specification. The same teaching style used for the first-year students is repeated, but at a more complex level. Our experience shows that most students can fulfill the assigned tasks, except a few of students whose ability in programming is low.

In the third year, a systematic course of approximately twenty hours on the SOFL formal engineering method is taught using the techniques described in Section 2. Much more students than those attending the first and second years' classes attend this class. Students seem to be more motivated to study this course, because many companies in Japan are interested in students who have studied new technologies. According to the data so far, an encouraging news is that almost all the students whose performance is within top 50% in this course could get satisfactory jobs in well-known companies and almost 100% of students who passed the examination of the course could find satisfactory jobs in industry.

In the forth year, all of the students who join SELDS under the first author's supervision will apply the SOFL method in modeling, inspection, and testing their software systems. Through these projects, students will learn more experience of using the SOFL formal notation and diagrammatic representation known as condition data flow diagram (CDFD). We usually organize the students into groups so that they will be able to experience how formal specifications can help in communication, cooperation, and documentation. All of the students who have graduated from SELDS found their satisfactory jobs as a system or software engineer in large enterprises, such as Sony, Fujitsu, NEC, CSK, etc. Although we have not heard of any serious case of formal methods being used in those companies so far, those students who are good at formal specification techniques are all working in responsible positions and play important roles in system analysis and design.

For postgraduate students in a master course or a Ph.D course, we teach formal refinement using the textbook by Morgan [8] so that they will learn how formal methods are expected to use during a development process. Since most of the students in the class of this course are not studying software engineering but other subjects, such as animation, graphics, artificial intelligence, network application, they usually face a difficult time. Since there is little hope to apply the refinement calculus in practice, students usually forget specific laws in the calculus later on but keep the fundamental idea of refinement in mind. We believe that the fundamental idea will help them make sensible decisions in practical software developments.

4. A SUCCESSFUL PROGRAM for TEACHING FORMAL METHODS to INDUSTRY

Many companies in Japan are the makers of some products, such as T.V sets, automobiles, trains, and traffic systems. To increase functions and reduce the cost in the upgrading and maintaining of systems, more and more functions are implemented by software. Since the failure of software in such an embedded system is likely to result in the disability of the whole system, causing inconvenience, frustration, or

even danger to customers and then a big economic loss to the makers, many companies begin to pay more and more attentions to formal methods. To introduce formal methods for use in companies, there are many difficulties to overcome, such as convincing company executives and engineers at various levels, educating people in formal methods, and ensuring constant technical supports for the application of formal methods in practice. It is hard to imagine the existence of a uniform solution to all the problems for all companies, but we have experienced a successful program to introduce the education of VDM to The Nippon Signal Co. Ltd. In this section, we present the major activities involved in the program.

The program consists of two stages. In the first stage, the three coauthors from the Nippon Signal have first successfully convinced the company executives that the adoption of VDM will create a great potential to improve the quality of systems and benefit the company, and then made a plan to teach VDM to engineers in all the relevant sections. According to the plan, some technical people from relevant groups are first chosen to learn VDM, and then those people will teach VDM to more people in their own groups. The essential idea of this plan is wonderful, but practically it could not progress as fast as we expected, because people who have learned VDM could not fully understand the motivation and effective ways to write formal specifications using VDM. To improve this situation, we established the collaboration between Hosei University and the company. The first task of the collaboration project is to review and discuss the original education plan. As a result, we worked out a more effective teaching plan in terms of schedule, contents, and teaching style. The first author also gave two seminars, on the basis of his previous experiences in both software engineering and formal methods projects in China, UK, and Japan, to the leaders and engineers of the relevant divisions in the company, in order to enhance the leaders and engineers' motivation and to help them build an accurate understanding of the real power of formal methods. The success of this stage has made us possible to proceed to the next stage.

The major activity in the second stage is to gain the support from the company executives to incorporate the teaching of VDM into the company's systematic education program. All the technical staff of the company can freely apply for the course, including both experienced and newly recruited staff. Each course takes about fifteen hours, and is taught within two days by the first author. The teaching techniques mentioned in Section 2 are used with flexibility. The surveys of all the courses taught so far have shown that over 50% of the students do not have a background in computer science or engineering, even some of them never experienced programming before. After the courses, almost 100% of the students become aware of the real role of formal methods in practical software development; more than 80% begin to understand how to write VDM specifications for real problems; and over 60% wish to use VDM in their future work.

The implementation of the above education program has led to the start of VDM applications in several groups at The Nippon Signal Co. Ltd. These applications have created a demand for effective software tool supports. We have begun a further cooperation to develop some supporting tools for VDM, such as automatic test case generation tool. We believe that our cooperation in both education and tools development will accelerate the process of transferring formal methods into industry.

5. CONTINUING EDUCATION

Our experience in applying and teaching formal methods in both academia and industry have convinced us that formal methods education is necessary and helpful, but it does not automatically mean that the teaching of formal methods is popular among students. Our observation shows that people with certain working experience usually find formal methods, particularly formal specification techniques, easy to learn and use, but this may not be true for students without working experience. The important reasons include that the students usually do not deeply understand the importance of the role of formal methods in software quality assurance and the contents of formal methods are quite complex. Since formal methods education is necessary and useful, a possible solution to this problem is to arrange formal methods as compulsory rather than optional courses. Thus, every student will be forced to learn formal methods. In addition to this assurance, by applying effective teaching methods, such as those mentioned above, and appropriate requirements for different level students, it would be highly possible to let more and more students learn formal methods. However, even if this possibility becomes reality, it will not guarantee that formal methods will become attractive to students. To be attractive, formal methods must achieve a good balance among the three qualities: simplicity, visualization, and preciseness, and must also demonstrate their benefits in ensuring software quality and reducing the cost of software projects. Teaching of formal methods must also provide fun for students, as in teaching computer graphics or animation. Unfortunately, few of existing formal methods have satisfied these criteria, and it is hard to imagine that any teaching method would significantly improve this situation. Since software development needs mathematical way of thinking, we believe that no matter whether formal methods are attractive or not, education in formal methods must continue at academia and hopefully in industry as well. Only education can make the application of formal methods in software engineering possible.

6. CONCLUSIONS

Education is the necessary and most effective way to transfer formal methods to software industry. The most important influence factor for the success of formal

methods education is whether the education is put in the context of software engineering. In this paper, we have described several techniques for teaching formal methods in the context of software engineering to both experienced and inexperienced students, each of which has been tested in practice. We believe that no matter whether formal methods can be used directly as an effective software engineering technique in practice, their education will definitely benefit software engineering practice through well-trained and well-disciplined engineers. The only way to effectively transfer formal methods to industry is: education, education, and education.

REFERENCES

[1] S. Liu. Formal Engineering for Industrial Software Development Using the SOFL Method. Springer-Verlag, ISBN 3-540-20602-7, 2004.

[2] S. Liu, A. J. Offutt, C. Ho-Stuart, Y. Sun, and M. Ohba. SOFL: A Formal Engineering Methodology for Industrial Applications. IEEE Transactions on Software Engineering, 24(1):337.344, January 1998. Special Issue on Formal Methods.

[3] B. P. Collins and C. J. Nix. The Use of Software Engineering, Including the Z notation, in the Development of CICS. Quality Assurance, 14(2):103.110, September 1988.

[4] D. L. Parnas. Inspection of Safety-Critical Software Using Program-Function Tables. In D. M. Hoffman and D. M. Weiss, editors, Software Fundamentals: Collected Papers by David L. Parnas, pages 371.382. Addison Wesley, 2001.

[5] S. Sahara. An Experience of Applying Formal Method on a Large Business Application (in Japanese). In Proceedings of 2004 Symposium of Science and Technology on System Verification, pages 93.100, Osaka, Japan, Feb. 4-6 2004. National Institute of Advanced Industrial Science and Technology (AIST).

[6] D. L. Parnas. Education for Computing Professionals. Computer, 23(1):17.22, 1990.

[7] C. B. Jones. Systematic Software Development Using VDM. 2nd edition, Prentice Hall, 1990.

[8] C. Morgan. Programming from Specifications. 2nd edition, Prentice-Hall, 1994.