Interpreting Beyond Syntactics: A Semiotic Learning Model for Computer Programming Languages

Jeffrey May
Computer Information Systems Department
James Madison University
Harrisonburg, VA 22807 USA
mayjl@jmu.edu

Gurpreet Dhillon
Information Systems Department
Virginia Commonwealth University
Richmond, Virginia 23284 USA
gdhillon@vcu.edu

ABSTRACT

In the information systems field there are numerous programming languages that can be used in specifying the behavior of concurrent and distributed systems. In the literature it has been argued that a lack of pragmatic and semantic consideration decreases the effectiveness of such specifications. In other words, to simply understand the syntactic features of a programming language alone does not provide an adequate foundation for students, programmers and designers to learn or to create robust and efficient programs. As a result, this paper will present a fresh approach for both teaching and understanding programming languages. The approach presented in this paper uses semiotics as a theoretical lens for identifying the important issues that transcend syntax issues alone and creates an organized conceptual model that will force instructors to facilitate a deeper understanding of programming constructs to their students.

Keywords: Programming Instruction, Conceptual Model, Semiotics, Language Construct Analysis

1. INTRODUCTION

For many college students, learning a new programming language and using that language to solve complex problems in an efficient manner can be a difficult or even seemingly impossible task (Robins et al., 2003). For introductory computer programming classes, student failure rates of greater than 50% are not uncommon. Similarly, it is not uncommon for organizations to spend countless education dollars on existing programmers to learn a new programming language only to find that many employees are not capable of making successful transitions from one language to the next (i.e. COBOL to Java). Thus, it could be argued that these high failure rates result from inadequate approaches to teaching new programming languages.

This paper argues that current approaches for teaching new programming languages tend to concentrate too much effort on syntactical issues and not enough organized or explicit attention to other important issues. Such a position is supported by Wing (1990) where she argues that a lack of pragmatic and semantic consideration results in poor programming output. Similar conclusions have been reached by Wiedenbeck and Ramalingam (1999) where they report that students from programming classes have a strong comprehension of program function, but are weak in their ability to comprehend deep-rooted issues such as control flow and other important efficiency-related issues. In other words, current programming instruction tends to create an environment where students focus their attention on the syntax of various programming constructs such as structure, shape, logic, and the appearance of the actual code without explicitly understanding other important issues such as efficiency, meaning, purpose, and proper usage of such code. Thus, Robins et al. (2003) concluded that a strategy for improving student comprehension of programming constructs needs to be explicitly presented to the programming community.

As a result, this paper will present such a strategy using a fresh and organized approach for learning new programming languages. The approach presented in this paper uses semiotics as a theoretical lens for identifying the important issues that go beyond syntax issues alone and creates a conceptual model that could be used in both universities and organizational settings as a learning tool. The goal of this

model is to create an organized approach for instructors that will force the explicit understanding of all surface level and deep rooted issues associated with various programming language constructs to their students. A student with a richer understanding of programming language constructs via an organized conceptual model may then be more capable of learning and adapting to the introduction of new languages in the future and will be more capable of creating elaborate and efficient programs.

To present such an approach this paper will first discuss semiotic concepts. This paper will then use semiotics as the governing theory to conduct a semiotic analysis in the context of learning a new programming language where the output of this analysis is an organized conceptual model.

2. SEMIOTICS

Semiotics is the study of signs where a sign is defined as anything that has meaning to somebody in some respect or capacity (Pierce, 1948; Stamper, 1973). More specifically, semiotics can be defined as the discipline that helps in studying information, information flow (communication), and culture. In other words, semiotics enables an accurate interpretation of meanings through acts of signification (Barley, 1983; Manning, 1992; Falkenberg).

The field of semiotics was first introduced by the American philosopher Charles Morris in 1901 where he introduced the notion that a sign can be broken down into three levels of abstraction known as syntactics, semantics, and pragmatics (Zemanek, 1966). In his book, *Signs, Language and Behavior,* Morris (1955) defined these three levels as: (1) *pragmatics* – deals with the origin, uses and effects of signs within the behavior in which they occur; (2) *semantics* – deals with the signification of signs in all modes of signifying; and (3) *syntactics* – deals with combination of signs without regard for their specific significations or their relation to the behavior in which they occur.

The semiotic ladder, an analytical tool that was later developed from the field of semiotics, represents the study of signs at six different layers of abstraction (Stamper, 1973; Liebenau and Backhouse, 1990). The semiotic ladder (Table 1) consists of physical, empiric, syntactic, semantic, pragmatic, and social layers. The six layers can further be classified into two levels - technical and human, pertaining to information flows.

Braf (2001) suggests the purpose of the semiotic ladder is to provide a framework for understanding the many different usages of information from technical to human considerations. The purpose of a programming language is to provide a means that allow communication of information between computers, from humans to computers, and also from humans to humans. Thus, learning a new programming language requires knowledge of both technical and human considerations. As a result, the semiotic ladder shown in Table 1 provides a theoretical tool that can be used to uncover the deep rooted and surface level issues that pertain to understanding how to learn and to use a computer programming language.

As with any framework, much confusion is often found when one attempts to completely distinguish between the individual layers. That is, many issues could be interpreted at more than one layer. Obviously, this confusion indicates that distinct boundaries between the layers of the semiotic ladder do not always appear for every design issue, especially when one is dealing with the human layers of the semiotic ladder (Kitiyadisai, 1991). However, one must attempt to find some place in the semiotic ladder for all the concepts and issues that pertain to the subject matter if one is planning to undertake a semiotic analysis. As a result, the remainder of this section will address the distinctions between the layers of the semiotic ladder shown in Table 1. These distinctions will be made in the context of an information system based on published research. Section 3 of this paper will then conduct a semiotic analysis in the context of learning a programming language using the Java language as an example.

TECHNICAL LEVEL
Physical Layer (Physical World) – signals, traces, physical distinctions, hardware
Empiric Layer – noise, entropy, pattern, variety, noise variety, redundancy, codes, efficiency
Syntactic Layer – formal structure, logic, data, records, files, computer language
HUMAN LEVEL
Semantic Layer – meanings, propositions, validity, truth, signification, denotation
Pragmatic Layer – communications, conversations, negotiations, intentions
Social Layer (Social World) – cultural norms, beliefs, expectations, commitments, culture, contracts, values, shared models of reality, attitudes

Table 1. Semiotic Ladder

2.1 Technical Level

The technical platform of any information system includes hardware, telecommunications, and software (Falkenberg et al., 1998). When dealing with these three components of any IS technical platform, Falkenberg et al. (1998) state that generally hardware maps to the physical layer, telecommunications maps to the empiric layer, and software maps to the syntactic layer of the semiotic ladder. These three layers require research mainly from the mathematics and natural science perspective and each individual layer of the technical level is well defined in the literature (Morris, 1964).

More specifically, the physical layer of the semiotic ladder is mainly concerned with modeling the properties of information as input to and output from any physical component of an information system. At the physical layer, the term information is generally defined as a collection of tokens that have both dynamic and static properties. A dynamic token is referred to as a signal and a static token is referred to as a mark. The physical layer is thus concerned with modeling these tokens in terms of their sources, destinations, and routes over which they are transmitted.

In contrast to the physical layer, the empiric layer views information in terms of its availability and usability. That is, the empiric layer is mainly concerned with the properties dealing with the transmission of tokens across channels of

communication. Clearly the engineering principles of noise, entropy, pattern, variety, noise variety, redundancy, codes, and efficiency would all be addressed at the empiric layer (refer to Table 1).

In contrast to the empiric layer, the syntactic layer is not concerned with any empirical or statistical properties of information; rather the syntactic layer is concerned with the form and shape of this information. That is, the syntactic layer is mainly concerned with the structure and form of tokens. This structure and form is generally expressed as syntax and requires generally agreed upon rules and formulations for consistency.

2.2 Human Level

Designers who limit themselves to only technical considerations are capable of understanding the importance of each technical layer and how they interact with each other. However, these layers generally disregard any human considerations. As a result, any designer who confines his attention to these layers alone typically will not address the business requirements and human considerations of an information system. As a result, information systems that are designed solely based on technical requirements generally fail (Dhillon and Backhouse, 1996). Hence, understanding the human level layers of the semiotic ladder is paramount to the success of any information system design.

As shown in Table 1, the semantic layer lies at the interface of the technical and human levels of the semiotic ladder. The semantic layer is mainly concerned with meanings and is not concerned with what language is used, how the message is encoded, or by what medium any message is transmitted on. Meaning can be interpreted by two very different semantic principles (Falkenberg et al., 1998). The first principle known as the objectivistic principle, assumes that meanings are mappings from syntactic structures onto objective features of a real world. The objectivistic principle assumes a perfect world scenario because it considers the real world to be the same for everyone and one that everyone knows independently of language. The second principle or constructivist principle has a more realistic orientation. It assumes that meanings are constructed and continuously tested and repaired. These repairs are made by the actions of people when using any syntactic structure. This involves an evaluation of language-action relationships. Of course, a well thought out semantic analysis would attempt to address and correct any potential failures before implementing any type of information system thus limiting the repair work that would need to be done after implementation. Semantic analysis would then consider various concepts such as propositions, validity, truth, signification, and denotation to uncover a rich understanding of meaning to all concerned with a particular information system.

In contrast to the semantic layer, the pragmatic layer is not concerned with semantic meaning; rather pragmatics is concerned with the intentions of both the sender and receiver in context. In other words, the pragmatic layer recognizes that meanings do not provide accurate or intended upon actions or reactions when taken out of context. At the pragmatic level, communication is studied intensively and is considered to be successful when a meaningful utterance is passed by a sender with a certain intention and is interpreted by the receiver of this utterance with the same intention. As a result, pragmatic analysis tends to deal with conversations, negotiations, and intentions of the social arena of an information system. Pragmatic analysis also helps in interpreting the patterns of behavior and obligation afforded by different stakeholders.

In contrast to the pragmatic layer, the social layer of the semiotic ladder deals with the consequences or outcomes of pragmatic communication. That is, when a meaningful utterance has occurred, the social layer would identify the social norms that would be changed, altered, or affected in some way. As shown in Table 1, some examples of these cultural norms might include: beliefs, expectations, functions, commitments, law, culture, contracts, values, shared models of reality, and attitudes.

3. A SEMIOTIC FRAMEWORK FOR ANALYZING PROGRAMMING CONSTRUCTS

The semiotic ladder shown in Table 1 provides a theoretical lens that can be used to uncover the surface level and deep rooted issues that pertain to various computer programming languages. Thus, this section conducts a semiotic analysis in the context of learning a new computer programming language. To conduct this analysis the conceptual model shown in Figure 1 was created by relating the semiotic concepts discussed earlier in this paper with the various nuances of learning a programming language. In other words, the conceptual model shown in Figure 1 was created by coupling semiotic concepts with the experience of the authors of this paper that has been gained from teaching various college level classes and professional seminars in programming. The goal of the conceptual framework shown in Figure 1 is to provide a conceptual model that forces a richer and more organized understanding of various programming languages and their associated constructs.

As shown in Figure 1, the conceptual framework is broken down into 2 separate types of analyses that include a physical layer analysis and multiple language construct analyses. The physical layer analysis is not required when dealing with the variety of language constructs offered by various programming languages. Hence, a physical level analysis is only required one time for individual programming languages. However, when dealing with specific language constructs, separate language construct analyses are required where each of these analyses require empiric, syntactic, semantic, pragmatic, and social layer investigations.

The remainder of this section will present the rationale behind the creation of the conceptual framework shown in Figure 1 along with illustrations of how to use it as a learning tool. It should be noted that the following analyses in this section will use the object oriented language of Java for illustration purposes. However, the conceptual framework presented in Figure 1 is intended to be scalable to all programming languages.

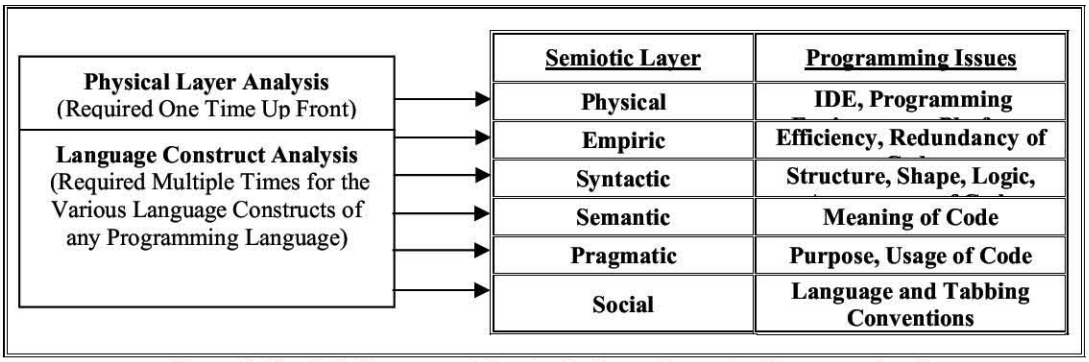

Figure 1. Semiotic Framework for Analyzing a Computer Programming Language

3.1 Physical Layer Analysis

As previously mentioned, physical layer analysis is mainly concerned with modeling the properties of information as input to and output from any physical component of an information system. In the context of computer programming, physical layer issues include: the type of environment, the actual integrated development environment (IDE) to be used, and the technical platform that the language supports. Obviously, physical layer issues should be addressed up front before attempting to understand language constructs.

For example when attempting to learn Java, Figure 2 illustrates the typical Java environment. As shown in Figure 2 information flow passes through a series of 5 steps.

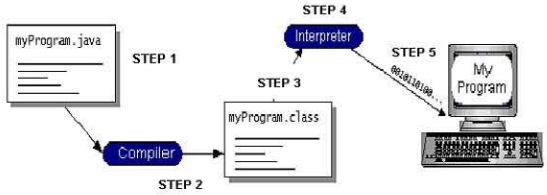

Figure 2. Java Environment

The 5 steps shown in Figure 2 are detailed in the following list:

1. **Source code generation.** Requires the selection of an appropriate Integrated Development Environment (IDE). For Java, various IDEs exist such as JCreator, JBuilder, Notepad etc. Obviously, a working knowledge of how to use the IDE of one's choice is important before attempting to apply any language. As shown in Figure 1, source code in Java is saved with a .java extension.
2. **Java Compiler**. In this step the source code is passed to the java compiler to be converted into bytecode. The result of compilation at this stage produces a binary file with a .class extension. All Java applications and applets are considered classes in the Java language.
3. **Java Interpreter**. With step three, the .class file is loaded into the Java interpreter. In order to do this, the entire contents of the .class file are placed into memory. Depending on whether one is making a Java application or applet, the class bytecode will be handled by the Interpreter for applications or by a Java Virtual Machine (JVM) and a browser for applets.
4. **Java Security**. With applets, the JVM runs a security check through the bytecode verifier to ensure that no language security rules are broken.
5. **Execution of Code.** For step 5, the bytecode is passed from the Interpreter to the local machine, one line at a time, as the interpreter parses the bytecode into machine language.

In terms of the technical platform, Java is considered platform independent as shown in Figure 3. That is, because local machines come equipped with a Java interpreter, code will run on any type of machine regardless of the type of operating system. Many other programming languages are platform dependent and require an upfront investigation of the stipulations of the language to be used versus that type of operating system.

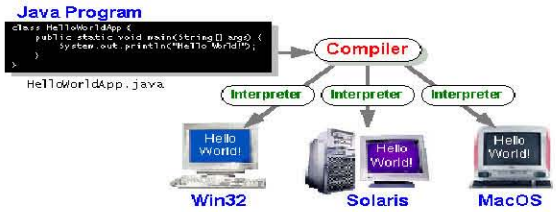

Figure 3. Java Technical Platform

3.2 Language Construct Analysis

When attempting to learn the vast array of language constructs that any programming language offers, this paper argues that a 5 layer semiotic analysis should be conducted. For the purposes of illustration, this paper will investigate the simple **if/else** selection structure of Java and conduct a 5 layer semiotic analysis using the conceptual model shown in Figure 1.

3.2.1 Syntactic Layer: As mentioned earlier, the syntactic layer is concerned with the structure and form of information. For computer programming, this structure and form is generally expressed as syntax and requires generally agreed upon rules and formulations for consistency. This paper argues that because the syntactic layer provides a visual of how a programmer can communicate with a computer, then syntactic layer analysis of any language construct should obviously come first. However, once syntactic layer analysis is complete, careful attention must be

given to the other 4 layers before one should attempt to write code using any particular language construct.

Most programming languages are organized by using basic syntactic structures for decision making (selection structures) and handling repetition (loops). However, the exact syntax across different programming languages usually differs to some degree. In Java, the syntax of the **if/else** selection structure is shown in Figure 4.

```
if (condition1 is true)          Example condition:
  statement1;                    if(x = = 3)
else if (condition2 is true)
{
  statement2;                    Example statement:
  statement3;                    System.out.println("hello");
}
else
  statement4;
```

Figure 4. Syntax of if/else Selection Structure in Java

3.2.2 Semantic Layer: The semantic layer is mainly concerned with meanings and is not concerned with what language is used, how the message is encoded, or by what medium any message is transmitted on. Hence, semantic layer analysis for programming language constructs consists of describing the meaning of how a particular syntactic structure works.

For the **if/else** structure in Java, several issues or rules must be addressed at the semantic layer. For example one must understand that when the computer encounters the code shown in Figure 4, it finds the first true condition and executes the statement or multiple statements associated with that **if** condition. However, once the first true condition is found and the statement or multiple statements are executed, the structure is no longer evaluated. If no condition is true, then the else condition executes.

At the semantic layer, the meaning of semicolons and brackets shown in Figure 4 require additional analysis. A semicolon is used to indicate that a statement is complete. And brackets are used to indicate membership. For example, *statement2* and *statement3* in Figure 4 only execute if *condition2* is true. Hence, these two statements belong to *condition2* and brackets are used to indicate this membership.

3.2.3 Pragmatic Layer: The pragmatic layer is not concerned with semantic meaning; rather pragmatics is concerned with the intentions of both the sender and receiver in context. In other words, the pragmatic layer recognizes that meanings do not provide accurate or intended upon actions or reactions when taken out of context. For programming language constructs the sender is the programmer and the receiver is the compiler or interpreter used by a particular language. Hence, the pragmatic layer is concerned with the purpose and usage of a particular language construct.

The purpose of the **if/else** selection structure shown in Figure 4 is to provide a means for decision making where an intended result or selection is made based on the truth of some type of condition. For example, if a program asks a user to enter a number that corresponds to a menu option, then the result of the user input will determine the result of the computer output. Since Java employs the use of Boolean logic, decisions must be made in the crisp domain of true/false.

3.2.4 Social Layer: The social layer is concerned with the social consequences or outcomes of pragmatic communication. That is, when a meaningful utterance has occurred, the social layer would identify the social norms that might influence the pragmatic layer or the social norms that might be changed, altered, or affected in some way as a result of the pragmatic layer. For programming language constructs, social layer analysis would mainly concentrate on how the social layer influences the pragmatic layer. Hence, the social layer would be concerned with the tabbing and language conventions of a particular language construct. That is, social layer analysis is concerned with the look and feel of programming code from the perspective of the human observer and is not concerned with how this impacts the computer.

For example, the code shown in Figure 4 could all be written on one single line with no negative consequences for the computer. However, as shown in Figure 4, the code is written in a modular manner where the individual pieces use only one line. This is done so that humans who read this code can understand it more easily. Additionally for readability purposes, if a statement can only be executed if a condition is true, then that statement or multiple statements should be tabbed over.

In the human world the notion of tabbing code is exactly the same as tabbing an outline for a written paper and is often times disregarded by newcomers as trivial. Yet, social layer analysis reminds us that if we as humans are going to communicate with each other via a particular programming language, then the social norms that dictate the way in which this language is written must be followed exactly.

3.2.5 Empiric Layer: As shown in Figure 1, the empiric layer is concerned with the efficiency and redundancy of code. Hence, empiric layer analysis attempts to find the most efficient manner in which to employ language constructs where enhanced computer performance is the ultimate goal.

For the **if/else** selection structure, efficiency concerns relate to using the **if/else** structure over using multiple **if** statements. For example, Figure 5 illustrates two separate but syntactically correct program statements that will do exactly the same thing. However if *condition1* is false, Code2 must evaluate 2 separate conditions whereas Code1 only has to evaluate 1 condition. Thus, Code1 is more efficient and will result in faster computer performance.

Code1: Efficient VS Code2: Not Efficient

```
if (condition1 is true)
  statement1;
else
  statement2;
```

```
if (condition1 is true)
  statement1;
if (condition2 is true)
  statement2;
```

Figure 5. Empiric Analysis of if/else Selection Structure

3.2.6 Summary of Language Constructs and Pedagogical Advice:

Table 2 illustrates the results of the language construct analysis conducted in this section using the **if/else** selection

structure of Java. As shown in Table 2, using the conceptual model shown in Figure 1 as a theoretical lens for investigating the **if/else** selection structure forces one to uncover and organize all of the pertinent issues that should be addressed before one attempts to employ the use of this particular language construct. Obviously, this same type of analysis could be conducted for the various other language constructs in Java such as the switch structure, repetition structures, methods, arrays, objects, inheritance, polymorphism, etc. And, this same type of analysis could be conducted for any other programming language. In other words, the conceptual model shown in Figure 1 is scalable across the various programming constructs of a particular language and across multiple languages.

To summarize, Table 2 provides a single instantiation of what might emerge via a language construct analysis as shown in Figure 1. Thus, an instructor could go through a similar exercise in the classroom where the finished product would be an organized table that easily allows students to truly have an enriched understanding of various language constructs. After one language construct analysis has been conducted by the instructor, students could then be asked to provide similar analyses of other constructs that could be turned in as part of an actual programming assignment. We argue that forcing this type of exercise on a student will facilitate a deeper understanding of programming constructs thus providing the foundation for constructing more elegant and efficient programs.

4. CONCLUSIONS

This paper argued that simply understanding the syntactic features of a programming language alone does not provide an adequate foundation for students, programmers and designers to learn or to create robust and efficient programs. Thus, many programming language construct issues that are certainly important may be overlooked when an individual attempts to employ these constructs for various program design scenarios. As a result, this paper investigated a fresh approach to learning programming languages. The approach presented in this paper used semiotics as a theoretical lens for identifying the important issues that go beyond syntax issues alone and created an organized conceptual model shown in Figure 1 that could be used in both universities and organizational settings to force instructors to facilitate a deeper understanding of programming constructs to their students.

This paper then investigated the simple **if/else** selection structure in Java to illustrate how the conceptual model shown in Figure 1 could be used as an explicit

Semiotic Layer	Language Construct Issues

1. Syntactic	• Visual representation of language construct
	if (*condition is true*)
	statement;
	else if
	(*condition is true*)
	{
	statement;
	statement;
	}
	else
	statement;
	Example condition:
	if(x = = 3)
	Example statement:
	System.out.println("hello");
	• Visual understanding of semicolons, brackets etc.

2. Semantic	• When encountering this code, the computer finds the first true condition and executes the statement or multiple statements associated with that if. • Once the first true condition is found, and the statement or multiple statements are executed, the structure is no longer evaluated • If more than one statement is associated with an if, then brackets must be used • Semicolons indicate the end of a statement
3. Pragmatic	• Used for decision making where decisions are made based on Boolean Logic (True/False)
4. Social	• If a statement can only be executed if a condition is true, then that statement or multiple statements should be tabbed over for readability purposes
5. Empiric	• Use if else rather than multiple if statements

Table 2. Language Construct Analysis using if/else Selection Structure

learning tool that would undoubtedly force a richer understanding of various programming language constructs. This richer understanding that can be brought about through the organized conceptual model shown in Figure 1 could certainly improve learning curves of individuals faced with learning a new pogramming language for the first time and could certainly increase an existing programmer's ability to create more elaborate and efficient programs. Additionally, because this approach is scalable across multiple languages, then it should follow that if one were capable of learning a first language using this approach, then one should be able to more quickly adapt to the nuances of any new languages thereafter.

5. FUTURE RESEARCH

To further extend the work of this paper, a number of future research opportunities exist. For example, a tool that identifies and is able to summarize the common practices used by a majority of programming instructors across a broad spectrum of universities should be created. Providing such information would then allow the research community to establish a baseline to compare the output of current approaches versus the modified approach presented in this paper. Thus, a further research question might then arise on how would we measure this output? Can we simply measure a student's comprehension of programming constructs via the programs they create or do we need to find alternative methods? In other words, is a program that does the job good enough?

6. REFERENCES

Barley, S.R. (1983), "Semiotics and the Study of Occupational and Organizational Cultures." Administrative Science Quarterly, Vol. 28, No. 3, pp. 393-413.

Braf, E. (2001), Knowledge or Information – What Makes the Difference? In Liu, K., Clarke, R. J., Andersen, P. B., Stamper, R. K. (eds.), Organizational Semiotics – Evolving a Science of Information Systems, Kluwer Academic Publishers, pp. 119-132.

Dhillon, G., and Backhouse, J. (1996), "Risks in the use of information technology within organizations." International Journal of Information Management, Vol. 16, No. 1, pp. 65-74.

Falkenberg, E., Hesse, W., Lindgreen, P., Nilsson, B., Oei, H., Rolland, C., Stamper, R., Van Assche, F., Verrijn-Stuart, A., and Voss, K. (1998), "A Framework of Information System Concepts." Laxenburg, Austria: International Federation for Information Processing (IFIP).

Kitiyadisai, K. (1991), "Relevance and information systems," Unpublished PhD Thesis, London School of Economics, University of London.

Liebenau, J., and Backhouse, J. (1990), Understanding Information. Basingstoke: Macmillan.

Liu, K. (2004), Virtual, Distributed and Flexible Organisations - Studies in Organisational Semiotics, Deventer, The Netherlands: Kluwer Academic Publishers.

Manning, P. (1992), Organizational Communication. New York: Aldine de Gruyter.

Morris, C. (1955), Signs, Language, and Behavior. G. Braziller, New York.

Morris, C. (1964), Signification and significance - a study of the relation of signs and values. Cambridge, Mass.: MIT Press.

Peirce, C. S. (1948), Collected Papers. Four Volumes. Harvard University Press.

Robins, A., Rountree, J. and Rountree, N. (2003), "Learning and Teaching Programming: A Review and Discussion." Computer Science Education, Vol. 13 No. 2, pp. 137-172.

Stamper, R. (1973), Information in Business and Administrative Systems. London, Batsford.

Wiedenbeck, S., & Ramalingam, V. (1999), "Novice Comprehension of Small Programs Written in the Procedural and Object-oriented Styles." International Journal of Human-Computer Studies, Vol. 51, pp. 71–87.

Wing, J.M. (1990), "A Specifier's Introduction to Formal Methods." Computer, Vol. 23, No. 9, pp. 8-24.

Zemanek, H. (1966), "Semiotics and Programming Languages." Communications of the ACM, Vol. 3, pp. 139-143.

Author Biographies

Jeffrey May is an instructor of Computer Information Systems in the College of Business at James Madison University and holds a PhD in Information Systems from Virginia Commonwealth University. Dr. May has taught programming courses in C++ and Java for 7 years and is currently teaching Business Statistics and introductory IS classes at JMU. His research interests include multi-objective decision analysis techniques, organizational IS security, and programming and logical design.

Gurpreet Dhillon is Professor of Information Systems in the College of Business at Virginia Commonwealth University, Richmond, USA and a Guest Professor at ISEG, Universidade Téchnica De Lisboa, Portugal. He holds a Ph.D. from the London School of Economics and Political Science, UK. His research interests include management of information security, ethical and legal implications of information technology. His research has been published in several journals including *Information Systems Research, Information & Management, Communications of the ACM, Computers & Security, European Journal of Information Systems, Information Systems Journal, and International Journal of Information Management* among others. Gurpreet has authored six books

including *Principles of Information Systems Security: text and cases* (John Wiley, 2007). He is also the Editor-in-Chief of the *Journal of Information System Security*. Gurpreet consults regularly with industry and government and has completed assignments for various organizations internationally.

Information Systems & Computing
Academic Professionals

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096