"Computer Science Majors' Perception of the Overlapping Cognitive Structures between Computer Programming and English Composition"

A Dissertation

Presented in Partial Fulfillment of Requirements for the Degree of Doctor of Philosophy in the College of Arts and Sciences Georgia State University

2004

by

Ruth Ann Goldfine

Committee:

[illegible]
Dr. George L. Pullman, Chair

Lynée Lewis Gaillet
Dr. Lynée Lewis Gaillet, Member

Mary E. Hocks
Dr. Mary E. Hocks, Member

July 19, 2004
Date

[illegible]
Dr. Matthew Roudané
Department Chair

Abstract

The purpose of this study was to determine (1) whether Computer Science (CS) and Computer Science Information Systems (CSIS) majors recognize the overlapping cognitive structures that exist in the areas of computer programming and English composition and (2) whether these students believe that the overlapping cognitive structures assist them in mastering the writing skills and strategies presented in a technical writing course. The 47 participants in this study were predominantly CS and CSIS majors at a large suburban university; although the class was open to all students at the university, very few participants were majoring in other disciplines.

Participants completed pre- and post-test self-evaluating questionnaires at the start and the conclusion of the academic semester, respectively, in order for the researcher to (1) identify their attitudes and perceptions regarding overlapping cognitive structures and the potential impact of this overlap on writing ability, and (2) measure whether participation in a writing course that did not explicitly draw attention to such cognitive structures had any impact on students' attitudes and perceptions regarding these structures. A select subset of students also responded to follow-up questions and/or participated in interviews.

The findings revealed that although instructors and researchers have identified overlaps in the cognitive structures used in programming and writing, and they have utilized these overlaps to teach English composition to CS/CSIS students, such overlaps are not readily apparent to students without prompting. Yet, these CS/CSIS students, when pressed, were able to identify some analogies between the programming and writing processes, though they professed disbelief that any such relationship between the two would be beneficial to them in a writing course. Consequently, in order for students to benefit from the overlapping cognitive structures that exist between writing and computer programming, instructors need to explicitly identify these structures to students and to develop assignments and activities that demonstrate how these overlapping cognitive structures lead to the transfer of knowledge from one discipline to the other.

Acknowledgements

First and foremost, I would like to express my deepest appreciation to Dr. George Pullman for his assistance, inspiration, and encouragement. Without his support, this research project would never have been realized. I also extend my gratitude to Dr. Lyneé Gaillet and Dr. Mary Hocks, both of whom provided insightful suggestions and comments that helped shape the direction of my research.

A tremendous debt of gratitude is owed my parents, Anthony and Patricia Rodak, who have always supported my educational pursuits and are largely responsible for my interest in academia. I am grateful, also, to my in-laws, the late Milton and Vivian "Jeep" Goldfine, for their extraordinary generosity in subsidizing the greater part of my graduate school expenses.

Most of all, a special heartfelt thank you goes out to my dear husband, Bernie, whose complete faith in me and assistance on the homefront gave me the confidence and time I needed to pursue a Doctoral Degree. Finally, to my twin boys, Will and Jake, who joined the family mid-way through my doctoral program: may my academic success be the inspiration for yours.

Table of Contents

List of Figures

Introduction

The development of college students' writing abilities, long the purview of English Departments, has, over the past decade, become a prominent issue in Computer Science departments at colleges and universities throughout the United States (Fell, Proulx, and Casey, 1996; Kaczmarczyk, 2003; Kay, 1998; Taylor and Paine, 1993; Walker, 1998). Although instructors in the discipline of computer studies have long recognized that writing is a fundamental activity in software development because programmers "must be willing to devote the necessary time and energy to refine written documentation" (Bickerstaff and Kaufman 44) and must be able to express their ideas clearly, covering all contingencies, since a computer only does what it is programmed to do (Fell, Proulx, and Casey 204), the recent emphasis on the writing capabilities of Computer Science (CS) and Computer Science Information Systems (CSIS) students covers a greater breadth of writing scenarios than simply programming and program documentation.

Specifically, a recent survey of potential employers of graduating CS/CSIS majors revealed that, in addition to technical expertise, these graduates are expected to possess strong written and spoken English skills (Pfeiffer 69, Hartman 32). Consequently, numerous Computer Science departments have focused on improving the oral and written communication skills of their students – with the intent of developing and applying these skills within the Computer Science program (Walker 24). This interest in students' writing capabilities has led to a profusion of presentations and articles regarding strategies for composition and, most notably, garnered significant attention at several recent Association for Computing Machinery (ACM) Special Interest Group on Computer Science Education (SIGCSE) Technical Symposiums on Computer Science.

Despite the ongoing conversation about the writing abilities of Computer Science majors, some researchers, such as Lisa Kaczmarczyk (2003), contend that

> currently there is little concrete evidence of progress towards increasing [computer science] students' writing ability, primarily because it has not been widely studied. A few educators have proposed tools [and] provided pedagogical tips or descriptions of classroom practice . . . . Some of this work has emphasized student benefits, but the perspective has been primarily Instructor-centered. This focus is problematic because successful teaching strategies depend upon understanding student perceptions. (341)

Kaczmarczyk's argument points to a need for student-centered research to determine the perceptions of CS/CSIS students and to parlay the findings of such studies into improved approaches for teaching writing skills and techniques to these students. The following study is an attempt to respond to the need Kaczmarczyk has identified by exploring the perceptions of CS/CSIS students participating in a technical writing course. The particular emphasis of the study is on whether these students identify any similarities between writing and computing programming and, if so, whether the students perceive these similarities to be of benefit to them in mastering the skills and techniques presented in a writing course.

<u>Purpose of the Study</u>

The purpose of this study was to determine (1) whether Computer Science (CS) and Computer Science Information Systems (CSIS) majors recognize the overlapping cognitive structures that exist in the areas of computer programming and English composition and (2) whether these students believe that the overlapping cognitive structures assist them in mastering the writing skills and strategies presented in a technical writing course. It was anticipated that an understanding of student perceptions regarding the influence of overlapping cognitive structures may inform composition theory, particularly as applicable to the teaching of technical writing, and influence instruction in the writing classroom.

Participants in this study were predominantly college juniors and seniors at a sizeable university (enrollment: 18,000). Most were CS or CSIS majors; all were enrolled in one of eight sections of an upper-level technical writing course taught by the English Department during the Fall 2003 and Spring 2004 semesters. Identical pre- and post-tests, which asked students to respond to statements using a Likert scale and solicited written follow-up comments to those statements, were administered at the beginning and end of the semester. The intent was to (1) determine students' perceptions toward the potential impact of their computing knowledge on their writing skills and (2) determine whether students' perceptions changed following their completion of the course.[1]

---

[1] The course was taught in its traditional style; that is, no attempt was made to point out similarities between natural and artificial languages, no analogies between programming and writing were discussed, and no discussion of interdisciplinary transference of skills was introduced.

Research Questions

Specifically, this study addressed and was guided by the following research questions:

Primary Questions

1. Do CS and CSIS students recognize the overlapping cognitive structures that exist in computer programming and in writing?
2. Do CS and CSIS students believe their understanding of computer programming assists them in mastering the writing skills and strategies presented in a technical writing course?

Secondary Questions

3. Should CS/CSIS students' perceptions of overlapping cognitive structures and the potential effect of this overlap on their writing abilities influence the way in which technical writing is taught to these students?
4. Do the results of the study have implications for the future of the technical writing curriculum?

Definition of Terms and Acronyms

The following terms and acronyms are central to this study:

Artificial Languages: For the purpose of this study, the term *artificial languages* designates the collection of computer programming languages employed to allow people and computers to communicate. Each language possesses its own vocabulary and syntax that is governed by a system of symbols and rules in order to ensure accurate and efficient communication between the programmer (human) and the computer (machine). Ambiguity is impermissible and will result in failed communication (i.e., the program will fail to run). Examples of these languages include Java, Visual Basic, and C++.

Coding: the process of linking new information to something that is already known

Cognitive Structures: interrelated or organized schemes (i.e., programs of action) for interacting with one's environment, developed through productive attempts at "handling, dismantling, and generally transforming [one's] surroundings" (Phillips and Soltis 42-43); derived from Piaget's studies of the cognitive development of children

Constructivism: a theory of learning which holds that knowledge is actively constructed by students who build recursively on the knowledge (i.e., facts, ideas, and beliefs) that they already possess

CS: Computer Science

CSIS: Computer Science Information Systems

Natural Language: For the purpose of this study, *natural language*, the naturally occurring communication system that exists among human beings, is limited to American English. Natural language includes an established vocabulary and syntax that are governed by a system of symbols and rules (e.g., punctuation), which ensure accurate and efficient communication between human beings. Ambiguity is ubiquitous but does not generally hamper understanding.

Transfer/Transfer of Training: the acquisition, performance, or relearning of a second habit because of, or as a result of the influence of, a previously established habit

<u>Assumptions</u>

The following assumptions are implicit in this research study:

1. The study participants responded honestly.
2. The instruments used in this research to gather data were sufficiently reliable and valid to allow accurate inferences with respect to the participants' knowledge, attitudes, and perceptions.
3. The research design and data analysis procedures were appropriate relative to the intent of this investigation.

<u>Limitations</u>

The limitations of this study are as follow:

1. Nearly all of the research participants were CS or CSIS majors in their junior or senior years at the university. Consequently, they are likely not representative of college students majoring in other disciplines, nor are they necessarily representative of first-year and sophomore students.

2. The pre-test administered at the start of the semester may have signaled the intent of the research to the participants, thereby causing them to alter their responses on the post-test to reflect what they believed was the desired or intended response.

Researcher Biases

The researcher was also the instructor who taught the sections of the technical writing course in which the participants were enrolled. Consequently, the researcher assumed dual roles, both as participant-observer and as teacher-researcher. In assuming these dual roles, the researcher placed herself "daily in the research and work environment . . . [affording herself] an insider, or *emic*, perspective on the research process" (Baumann and Duffy-Hester 2). Some scholars have argued that ethical concerns may arise when researchers take on dual roles (Dorsel, 1981; Hammack, 1997) – such as the potential for intentional or unintentional coercion to enlist student/subject participation and the possibility that the researcher might face a choice between either meeting the needs of the students or satisfying the demands of the research. However, Bissex and Bullock point out that "research methods are not neutral tools; they embody assumptions about causation and control, about how knowledge is acquired, and about the researcher's relationship to what is being studied" (12); therefore, the challenge of maintaining neutrality is not unique to the teacher-researcher but exists in some form in most types of research.

In its traditional form, research is seen as the purview of the detached data-gatherer churning out objective, factual data; however, "objectivity is not the sole route to knowledge. There is knowledge of a different sort to be gained through empathy and involvement, through sympathetic observation that seeks to understand the experience of other persons rather than their behavior as objects" (Bissex and Bullock 12-13). Kelly Chandler, in her article "Working in Her Own Context: A Case Study of One Teacher Researcher," notes that the relationship between a teacher and his or her students allows for meaningful insights on very specific issues (30). In particular, she points to the experiences of teacher-researcher Lorna Tobin, who commented, "When I felt things weren't going the way I thought they should (whatever way that was), I went to the kids for help in answering my questions" (27).

The teacher-researcher phenomenon is highly accepted in the area of rhetoric and composition research; in fact, researchers' assumption of dual roles is not only a respected – and perhaps expected – practice but can yield results and insights that would be otherwise impossible. In particular, practitioners and proponents of qualitative research and naturalistic inquiry argue the value of relationships such as those that exist between teacher and student, noting that such involvement allows the researcher to empathize with the subjects to the degree that he or she is able to elicit personal stories or in-depth descriptions (Rubin and Rubin 13). Lincoln and Guba devote an entire chapter in *Naturalistic Inquiry* to establishing the trustworthiness of data derived from the type of qualitative, naturalistic research inherent in a study in which a teacher uses his or her own students as subjects (289-331). Furthermore, the teacher-researcher phenomenon can eliminate (or at least ease) difficulties in gaining access to subjects, a concern discussed by scholars such as Erlandson et al. (1993) and Stake (1995).

Additionally, as a participant-observer in this study, the researcher's findings were shaped by numerous factors that are uniquely her own:

- her role as both instructor and researcher at a university,
- her education and experiences in academia while pursuing undergraduate and graduate degrees,
- her understanding of and abilities in the use of artificial languages gained through research and coursework, and
- her comprehension of the writing process as acquired in her graduate studies and her experience as an English instructor.

Given these factors, the researcher, particularly in interpreting the quantitative data and making meaning out of the qualitative responses, imposed a reality distinctly her own.

Although it is arguably impossible for any researcher to identify all of her biases at the outset of a study or to control the possibility of new ones emerging during the course of a study (Lincoln and Guba 282), the researcher of this study acknowledges the following biases:

- a belief in the social construction of knowledge, and
- a belief in constructivism – a theory of learning by which students are believed to rely on their existing knowledge to make sense of new concepts.

Importance of the Study

The tremendous interest of Computer Science departments in the writing capabilities of their majors over the past several years, which echoes – or perhaps results from – the concerns and/or expectations of employers, gives testament to the desire of educators in the field of computer studies to ensure their graduates are skilled in the art of written communication. Despite this obvious desire to promote writing skills among CS majors, Lisa Kaczmarczyk contends that there is inadequate evidence of progress in this area because of limited research. In particular, Kaczmarczyk points to an absence of a student-centered perspective – a significant omission that she believes has hindered the development of successful teaching strategies (341). Conducting such student-centered research may inform pedagogy and, in turn, result in improved writing among CS/CSIS majors.

This study is, in essence, an attempt to fill the research gap noted by Kaczmarczyk. The goal of this study was to take a first step toward investigating CS/CSIS students' perceptions of writing, particularly regarding these students' recognition of the overlapping cognitive structures in programming and writing, and regarding their attitudes toward the significance of these overlaps in helping them to master writing skills and strategies. It was hoped that such research would provide valuable information to assist in tailoring an approach to writing that meets the specific needs and expectations of CS/CSIS majors by best utilizing their unique talents, skills, and existing knowledge base. Although the findings of this study are specific to CS/CSIS students participating in a technical writing course, it is hoped that they will have a broader applicability, perhaps serving as the blueprint to guide researchers in all disciplines in identifying and capitalizing upon students' perceptions and existing knowledge by tailoring courses such that they promote student learning by building upon the knowledge students already possess.

Organization of the Remaining Sections

The remainder of this document is divided into the following four sections:

- Review of the Literature: a discussion of the available literature relevant to this study.
- Methodology and Procedures: a description of the setting, participants, instruments, and data gathering procedures.
- Description and Discussion of the Data: a presentation and explanation of the data gathered during this study.
- Findings and Key Recommendations: an interpretation of the data, emphasizing conclusions and recommendations for future research.

Review of the Literature

The following review of literature resulted from an extensive search of numerous interdisciplinary online databases (such as Access Science (McGraw-Hill), Research Library from ProQuest, Academic Search Premier at EBSCOHost, and ISI Current Contents) and a wide variety of professional journals (for example, *AI Magazine, Annual Review of Psychology, College Composition and Communication, College English, International Review of Applied Linguistics in Language Teaching, Journal of Advanced Composition, Linguistics and Philosophy, Modern Language Journal, Technical Communication Quarterly, Research in the Teaching of English, Teaching English in the Two-Year College,* and *Technical Communications).* A search of these databases and journals produced no previous studies that addressed the specific topic of this investigation. Therefore, five key areas deemed to be instrumental in achieving the goals of this study were researched to provide the foundation for accurately interpreting and understanding the data gathered:

- theories of learning and composition theory[2]
- natural language
- artificial languages
- comparison of natural language and artificial languages
- CS/CSIS majors in the writing classroom

[2] Though these are technically two separate topics, a combined discussion of the two was deemed the most efficient and appropriate way to accommodate their closely intertwined ideas and concepts.

Theories of Learning and Composition Theory

Although this study focuses narrowly on computer programming and writing, in a broader sense, it deals with theories of learning. A current and – arguably – dominant theory of learning is *constructivism*, which claims that knowledge is actively constructed by students who build recursively on the knowledge (i.e., facts, ideas, and beliefs) that they already possess. That is, the constructivist theory of knowledge "recognizes that it is the learner who constructs knowledge, not the teacher who imparts it" (Biggs 2). Two tenets of constructivism are (1) a teacher cannot ignore a student's existing knowledge but rather must question the student in order to understand what models the student possesses and only then attempt to guide the student on the correct theory and (2) sensory data combines with existing knowledge to create new *cognitive structures*, which are in turn the basis for further construction (Ben-Ari 257-258).

Constructivism echoes the concepts put forth by earlier researchers (e.g., Biggs and Moore, 1993, who build on the work of Piaget; Collins and Quillian, 1990; Donnelly 1994; and Phillips and Soltis, 1985, who draw on the research of Bruner) who studied cognition, learning, and memory. These researchers argue that not only does most learning proceed by "building new structures on the basis of preexisting structures, not by forming arbitrary new associations" but also that "how fully a person understands a sentence – or any experience he has – depends on how much stored information he relates to it" (Collins and Quillian 119). Similarly, research in the area of linguistics determined that "when we approach any text, our expectations, prior knowledge, and individual experience all affect our ability to process that text" (Donnelly 30). For example, Biggs and Moore describe the use of coding, the process of linking new information to something that is already known, as a means of remembering new information. In their example, an individual was asked to remember a series of numbers by linking it to an already known series of numbers, such as a telephone number, street address, or Social Security number:

> Coding takes place on the basis of previous knowledge. But how can we code a new experience when nothing quite like it has occurred previously? It all depends on how much the new experience has in common with earlier ones. If there is a very great deal in common, we might code the new experience as "Uh-huh. Thingamajig again" . . . . Or we might note some differences and, in trying to make sense out of them, recombine our past experience in new ways, as when trying to work out the meaning of a new word from its context. That recombination is called *recoding*. (215-216)

Biggs and Moore further theorize that each new experience is matched to what is already known and that four basic "matching" possibilities exist, each with different and important cognitive and motivational consequences: (1) no mismatch: the match is exact (or near enough); recognition and coding occur, but no rethinking or recoding; (2) some mismatch: enough to challenge but not overwhelm, recoding is necessary but easily achieved; (3) much more mismatch: cannot be handled, desperate recoding, panic; (4) all mismatch: no comprehension, tune out, rely on non-verbal ways to reach goal (216).

Biggs and Moore's theories derive from the work of Piaget. Piaget argued that learners interact with their environment, drawing upon the cognitive structures they already possess to make sense of the current experience. If the experience is one that has been engaged in many times before, the learner will be able to deal with it satisfactorily. It is this argument, in part, that suggests to some researchers, such as Jerome Bruner, that prior experience plays a significant role in education (Phillips and Soltis 45-49). In 1959, Bruner, then a psychologist at Harvard, issued a report in which he discussed how students could be prepared by their learning *now* to tackle problems in the *future*. He argued that students must grasp the *structure* of the discipline if they are to accomplish this feat:

> Grasping the structure of a subject is understanding it in a way that permits many other things to be related to it meaningfully. To learn structure, in short, is to learn how things are related. . . . in order for a person to be able to recognize the applicability or inapplicability of an idea to a new situation and to broaden learning thereby, he must have clearly in mind the general nature of the phenomenon with which he is dealing. (Phillips and Soltis 56)

At the heart of the theories espoused by Biggs and Moore, Piaget, and Bruner is the notion that learners, indeed all humans, will face new, unknown experiences and challenges throughout their lives. Some of these challenges may include familiar elements; others may be totally dissimilar to any of the cognitive structures the learners possess. Yet, the majority of people successfully negotiate such challenges by relying on the cognitive structures that they have. How? Through a phenomenon known as transfer.

Robert Travers, in *Essentials of Learning* (1967), asserts that

> all teaching is based on the assumption that the immediate skills, understandings, attitudes, appreciations, and other learned functions influence behavior in a diversity of subsequent situations. Skill in the writing of English is not taught so that the pupil may write better themes in school, but so that he will be able to prepare all kinds of effective written communications in his daily life. It is assumed that there will be *transfer of training*[3] from whatever is learned in the field of writing in school to whatever has to be written in outside situations. (234)

It would seem, then, that at its very core, education promotes, whether implicitly or explicitly, not only discipline-specific knowledge, skills, and capabilities but also the capacity to recognize situations that are similar to previous ones and the ability to draw upon and apply the necessary (i.e., relevant) cognitive structures. In fact, Travers asserts that "all learning takes place within the context of previous learning and hence involves transfer" (235). He further claims that students must learn to code whatever it is they have learned as belonging to a class. It is learning to code situations as presenting or not presenting the essential features of an existing cognitive structure that enables transfer; the application of a principle (i.e., cognitive structure) to different situations "demands

[3] "Transfer of training, broadly defined, occurs whenever the existence of a previously established habit has an influence upon the acquisition, performance, or relearning of a second habit" (McGeogh and Irion, qtd. in Travers 235).

that the situations have some similarity, but the similarity may be at an abstract level" (253).

In the 1970s, researchers such as M. C. Wittrock developed a *generative hypothesis* that interpreted learning primarily as "the construction of concrete, specific verbal and imaginal associations, using one's prior experience as part of the context for the construction. It is a model of learning as the transfer of previous learning" (173). That is, when learners relate new information to their experience and are required to construct associations or meaning involving the new information, their learning and recall is facilitated (173). Likewise, other researchers (e.g., Gagné 1962; Bell-Gredler, 1986; Gagné and Perkins-Driscoll 1988) believe that productive learning is

> a matter of transfer of learning from component learning sets to a new activity that [includes or incorporates] previously acquired capabilities. The implications [are] that learning is cumulative and intellectual development may be conceived as the building of increasingly complex and interesting structures of learned capabilities. (McClendon 32)

Students' use of cognitive structures and the phenomenon of transfer of training, key in the constructivist theory of learning, are also significant in composition theory, particularly in Flower and Hayes' cognitive theory of composition (1981, 1988). Flower and Hayes believe that a writer in the act of writing is "hard at work searching memory, forming concepts, and forging a new structure of ideas, while at the same time trying to juggle all the constraints imposed by his or her purpose, audience, and language itself" ("Cognition" 92), and they contend that the writer has, in his or her long-term memory, stored knowledge "not only of the topic, but of the audience and of various writing plans" ("Cognitive" 369). Reading and writing theorist Sally Barr Reagan echoes this emphasis on stored knowledge, claiming that "the greater the variety of texts readers encounter, the wider their knowledge, which carries over and is applied when they begin to write" (179). Flower and Hayes, as well as Reagan, have successfully extended the constructivist theory of learning to the specific discipline of composition studies, and they have done so by referencing stored memories in broad terms. Therefore, it seems possible that the concepts of constructivism could be applied successfully to an investigation of the impact of interdisciplinary stored knowledge – that is, cognitive structures – on the ability of students (i.e., specifically, CS/CSIS students for the purpose of this study) to master writing skills and strategies.

Flower and Hayes also contend that writers attempt to build a coherent network of ideas (i.e., to create *meaning*); as evidence of this, they cite writers' attempts to "test or evaluate what they've just said to see if it is related to or consistent with other ideas" ("Cognition" 98). This notion is further supported by Frank Smith, who in his book *Understanding Reading: A Psycholinguistic Analysis of Reading and Learning to Read* claims that humans can only make sense of the world in terms of what they already know (8). The implication of his claim, as regards this study, is that perhaps CS/CSIS students, in referring to their stored knowledge of programming, make connections or "make sense" of composition by relying on the interdisciplinary knowledge they already possess, specifically, their knowledge of how to "write" (i.e., program) using artificial languages.

Furthermore, Smith states that our memories (i.e., our accumulated knowledge) are not merely a collection of assorted snapshots, videos, and tape recordings of bits of our past. Rather all our memories have meaning in that they are related to everything else we know, more like a summary of our past experience. Specifically, he states: "I do not want to remember that on 16 July I sat on a chair, and that on 17 July I sat on a chair, and on 18 July I sat on a chair. I want to remember that chairs are for sitting on, a summary of my experience" (Smith 8). His conclusion suggests that humans have the capacity to make associations that allow for the extension of skills and knowledge beyond the specific situation during which and for which they were initially acquired. Perhaps, then, writers draw on skills and knowledge obtained outside the writing classroom to aid them in understanding and mastering the writing process. For example, a college student who writes home telling his parents about the time he spent studying in the library Saturday afternoon but chooses not to tell them about the party he went to Saturday night clearly understands the importance of audience in determining what to write. Similarly, anyone who has built a model, followed a recipe, or programmed a VCR can appreciate the need for logic and structure to ensure clarity. However, whether writers tap into the knowledge and skills acquired outside the writing classroom to assist them in their composition tasks may not be readily apparent if these writers have attained a relatively high degree of competency in writing. Flower and Hayes anticipate that as writers become more experienced, many of the basic goals associated with writing (e.g., starting with an introduction) become automated to the point that these objectives are not even consciously considered by the writers ("Cognitive" 381). Consequently, well-learned skills from other disciplines that are applied to the act of composing may flow so naturally for some writers that they are not even aware that they are applying the knowledge they have gained from an "outside" source.

Flower and Hayes also acknowledge that extensive reading affects a person's ability to write. Specifically, a well-read person is more likely to be a good writer than a person who is not well-read, simply because the former has a much "larger and richer set of images" of what a text can look like ("Cognition" 99). This finding is further supported by the reading and writing approach to composition. Reading and writing theorists such as Birnbaum (1986), Reagan (1986), Sternglass (1983, 1986), and Tierney (1986) demonstrate that the skills of reading and writing both employ similar cognitive processes and rely upon a common text knowledge; therefore, experienced readers are usually proficient writers, while inexperienced readers are almost always basic writers (177). These findings support the constructivist theory of knowledge transfer from existing cognitive structures to the mastery of a new challenge or skill.

While the implication seems to be that transference may occur equally and without bias from any discipline or experience to any other discipline or experience, of particular interest in this study is the transfer of knowledge from the cognitive structures acquired in the discipline of Computer Science – particularly in the area of computer programming – to the study of English composition. The processes the disciplines use to create their end-products are strikingly similar; each includes audience analysis (i.e., end-user/reader), design (i.e., pseudo-coding/outlining), review (i.e., user testing/peer review), and chunking (i.e., subroutines/paragraphs and subsections), to name a few.

Additionally, at the most basic level, computer programming and composition are incredibly similar: both require that developers (i.e., programmers and writers) use a formal language – whether artificial or natural – that includes a specific vocabulary, syntax, and punctuation to develop an end-product (i.e., computer program or written document) that meets the audience's (i.e., user's or reader's) needs. Given the similarities between programming and writing, upper-level CS/CSIS majors may in fact possess a wellspring of valuable cognitive structures, developed through their training in computer programming, that overlap those structures invoked in English composition. Specifically, since both programming and writing require that students "write" using a formal language, an exploration of both natural and artificial languages was deemed essential to this study in order to better understand potential overlaps in the cognitive structures of the two disciplines.

Natural Language

"Natural language" refers to the naturally occurring communication system that exists among human beings. Specifically,

> when you speak English – or any other language – you are using a system of sounds that have developed and evolved over a long period of time. The language you learned growing up is called a *natural language*. In other words, it is not an artificial language or one made up by humans for computers, machines, or some special purpose. Natural human languages

> are very technical and governed by rules . . . [but] are sensitive to people and the communities they live in. (Ellis 1)

More simply stated, natural language is "a system of symbols and rules that enable us to communicate," where words, either written or spoken, are symbols and the rules specify how those words are ordered to form sentences (Harley 2). Additionally, all languages have vocabulary and syntax that, when combined with the practical rules of language use, constitute a grammar – that is, everything that is known about the workings of a language (Ellis 14).

Early myths regarding language, including the "Genesis" story in the *Bible*, simply stated that language existed and a god gave it to man. Plato, on the other hand, although he accepted the facts of language as given, questioned how language came about and wondered about the principle that guided the creation of the first words. Plato's approach, radical for its time, paved the way for future language exploration, resulting in the modern view that language is the result of a natural evolutionary process and the natural influences that affect the development of humans (Ellis 3-4).

Borrowing from Harvey Daniel's (1983) book *Famous Last Words: The American Language Crisis Reconsidered*, Donald Ellis asserts that numerous claims about natural language are accepted by all language theorists (13). Of particular relevance to this study are the following three claims:

- *Language is rule governed.* To learn a language, one must learn a vast system of rules. These rules, which govern sounds, words, the arrangement of words, and the social aspects of speaking, are largely subconscious and can be applied without being understood. The rules are arbitrary; words can change meanings, and rules of sentence structure are different in different languages (13).

- *All languages have sounds, vocabulary, and syntax.* Sounds are the inventory of human noises that become meaningful; these are then organized into vocabulary, which represents ideas, things, and actions in the world. Syntax is the organization of words (i.e., the vocabulary) into sentences to represent relations among ideas. These three components, combined with the practical rules of language use, constitute a grammar (14).
- *Writing is derived from speech.* Writing, a derivative form of speaking, is based on a set of visual conventions (i.e., the alphabet) that represent the sounds of speech and is subject to many rules and variations. The development of writing, beginning about 5,000 years ago, "was the first baby step on the journey to computers, databases, and massive information dissemination" (16).

Mastery of the grammar of natural language perhaps gains significance when the written word, as opposed to the spoken word, is being used to communicate. In order to ensure accurate and efficient communication, both writer and reader must be knowledgeable of the writing system for their language. That is, they must know

> the regularities of the written language, the forms of written symbols, the way they are sequenced on the page, the system of punctuation marks and

> other typographical conventions, the systems of spelling in sound-based written languages, a set of individual idiosyncratic word forms, and much else beside. (Cook 1-2)

Although natural language is governed by specific rules, sufficient flexibility exists within those rules to allow for ambiguity. For example, all languages adhere to syntactical rules of one sort or another, but the rigidity of these rules is greater in some languages. In the English language, syntax is especially important because meaning is determined by word order. In the sentence "The friend of the mother and the father will arrive soon," it is possible to attach two different meanings to the sentence. One person – the friend of the mother and father – may be arriving soon, or two people – the friend of the mother and also the friend of the father – may be arriving soon. Such ambiguity is called structural ambiguity because it arises from the fact that there is more than one way for the word elements to combine into meaning (Ellis 29-31; Harley 308).

Similarly, punctuation plays a significant role in ensuring the comprehension of written language. Punctuation divides information into meaningful chunks and, in effect, serves as guideposts to the reader. Punctuation has always been inherently systematic. Although centuries ago each scribe may have used a different system, the punctuation system underwent major developments at periods (roughly, the fourth and seventeenth-eighteenth centuries) when literacy was expanding across class boundaries and writers could no longer assume as much common mental ground with readers as they had previously taken for granted (Mann 362-363).

In the teaching of punctuation, Nancy Mann advocates drawing upon analogies with other disciplines. For example, she claims that the easiest of the nonessential markers to teach is the colon because instructors can capitalize on students' familiarity with the equals sign by using math symbols to represent sentences. Thus, the common error of placing a colon between the verb and a list of objects (as in *We demand: A, B, and C)* is like an equation with no term on the left-hand side, while placing a colon between the verb and a list of complements (*My reasons are: 1, 2, and 3*) is like using two equals signs in a row (384).

Syntax and punctuation are essential because natural language is not a formal system (or logistic system) in which a sign (i.e., symbol/word) has only one meaning. Every element of language – from phonemes (invariants of the sounds of speech), to morphemes, to words, to sentences, to text segments – can have multiple meanings; this is simply the way the human mind works. On the contrary, artificial languages (i.e., computer programming languages) are able to operate only within a system in which each sign has a single meaning (Kreymer 2). Despite this significant difference, natural and artificial languages actually have quite a bit in common.

Artificial Languages

The term "artificial languages," for the purpose of this paper, designates the collection of computer programming languages employed to allow people and machines (specifically, computers) to communicate. Computer programming languages, then, can be defined as "the different notations used to communicate algorithms to a computer [where] an algorithm is the set of instructions and the order in which they have to be performed" ("Programming" 1). Notably, computers can only understand instructions presented in machine language, a sequence of binary numbers (zeroes and ones) where each number may indicate the operator (i.e., instruction to be executed) or the operands (i.e., the pieces of data) on which the instruction is performed. Programming in machine language is cumbersome and tedious; however, the earliest programmers had no other options.

In the 1940s, programmers of the first computers had no choice but to write instructions in sequences of binary digits (i.e., machine language) that the computer could understand and execute. Instructions were written in an eight-digit format WWXYYYYY, "where WW stands for an arithmetic operation, X signifies a register (called an operand register for performing arithmetic calculations) to contain one argument for an operation, and YYYYY signifies the memory location" ("Programming" 1). Understandably, programming in this manner proved difficult and resulted in numerous errors.

To facilitate programming and alleviate errors, assembly language was developed to replace machine language. In assembly language, all locations were provided easy-to-remember names and machine instructions were given symbolic names. A relatively simple program (i.e., an assembler) converted this symbolic notation into an equivalent machine language program. Though assembly language was an improvement over machine language, programs were still very hard to write and mistakes were common because programmers were forced to think in terms of the computer's architecture rather than in the domain of the problem being solved.

In the late 1950s, the first higher-level programming languages were patterned after mathematical notation. These languages were based on the concept that to compute $|A + B - C|$ and store the result in a memory location called D, all programmers had to do was write $D = |A + B - C|$ and let a computer program, the compiler, convert that equation into the sequences of numbers that the computer could execute. FORTRAN (an acronym for Formula Translation), developed by IBM under the leadership of John Backus in 1957, was the first major language in this period. FORTRAN statements were patterned after mathematical notation; therefore, the above example would be written as D = abs (A + B – C) where *abs* represented a function that computed the absolute value of its argument. The basic program unit is the subroutine or function. Each subroutine is a self-contained unit that computes some value. Notably, FORTRAN became the model for most current languages, including BASIC, C, C++, and Java ("Programming" 4-5).

Initially, there was resistance to higher-level languages (called higher-level because they were deemed more complex or "higher" than the assembly languages of that day). Computers were extremely expensive, and early compilers often generated inefficient sequences of instructions in compiling a program. Furthermore, although programmers made mistakes in assembly code, it was still deemed less expensive and more efficient to correct those errors than to run ineffective programs on these expensive machines. However, higher-level languages are viewed quite differently in the 21st Century. Computers have become significantly less expensive and substantially faster;

moreover, compiler technology has advanced to the point that compiled programs often run as fast or faster than handwritten assembly code. Among these higher-level languages, several have made lasting contributions to artificial language design: FORTRAN, LISP, ALGOL, COBOL, C, C++, and Java. Moreover, almost all modern concepts of computer programming languages first appeared in the four important languages developed from 1957 through 1962: FORTRAN, ALGOL-60, COBOL, and LISP ("Programming" 5).

Around the time FORTRAN was being designed, John McCarthy at the Massachusetts Institute of Technology was investigating software for problem solving. McCarthy developed LISP (for LISt Processing), an applicative programming language that is the basis for almost all artificial intelligence and expert system developments since then ("Programming" 5-6). In 1960, an international committee led by Peter Naur and John Backus, developed ALGOL-60[4] by extending the ideas present in FORTRAN. A notation called BNF (Backus-Naur Form), a grammatical form called a context-free grammar, defined the language; this grammar, coupled with advances in compiler technology, allowed efficient compilers to be developed almost automatically. Additionally, ALGOL-60 further developed the concept of the subroutine (called a procedure in ALGOL-60) to permit the easy communication of data from the calling procedure to the called procedure ("Programming" 5).

[4] ALGOL-60 is an acronym for ALGOrithmic Language, 1960.

Also in 1960, the U.S. Defense Department, under the leadership of Grace Hopper, organized the development of COBOL[5]. COBOL introduced the concept of record to provide access to data for business applications. During the 1970s, Niklaus Wirth further refined this concept with the type declaration and the record type in the Pascal programming language. Also during the early 1970s, AT&T Bell Laboratories developed C, a higher-level computer language with a structure like FORTRAN. A C program consists of several procedures, each composed of several statements, including the IF, WHILE, and FOR statements. However, a primary focus of C was to include operations that allow the programmer access to the underlying hardware of the computer. Therefore, C includes a large number of operators to manipulate machine language data in the computer and a strong dependence on reference variables so that C programs are able to manipulate the addressing hardware of the machine ("Programming" 5, 24).

Following C was C++, developed in the early 1980s as an extension to C by Bjarne Stroustrup at AT&T Bell Labs. The idea was to extend C with Smalltalk-like classes. Each C++ class would include a record declaration as well as a set of associated functions. In addition, an inheritance mechanism similar to Smalltalk was included in order to provide for a class hierarchy for any program ("Programming" 24).

By the early 1990s, the World Wide Web was becoming a significant force in the computing community, and web browsers were becoming ubiquitous. For security reasons, browsers were designed such that they could not affect the disk storage of the machines they were running on. All computations performed by a web page were carried

---

[5] Common Business Oriented Language

out on the web server accessed by web address (i.e., its Uniform Resource Locator, or URL) in order to prevent web pages from installing viruses on user machines or inadvertently (or intentionally) destroying the disk storage of the user. Consequently, all information had to be transferred from the user to the web server.

During this period, James Gosling at Sun Microsystems led the development of a product called Oak – which evolved into Java in 1995 – that could execute on the user's machine. Instead of transferring all of the user's data to the server machine, the server would transfer the application program to the user. In order to make this transfer efficient, the transferred program had to be both small and capable of running on every browser on every computer connected to the Internet. Sun decided to transfer a version of the source program (the byte code) rather than the usually longer machine language program. A program built into the browser executed the byte code as if it were a machine language program for that particular machine. Such a program is typically called an interpreter.

The constraint on this byte code was that it could not affect the disk storage of the user's machine. Initially, C++ was the programming language of choice due to its class hierarchy and efficient execution model. However, the pointer references and binary operations on machine addresses in its underlying C base language made it impossible to assure adherence to the web security model. Consequently, C++ was dropped as the base language and was replaced by Java, which bears a strong similarity to C++ without many of the problems of C++ ("Programming" 25).

Keywords (sometimes called *reserved words*) are the built-in vocabulary of Java. Often these keywords express a command or an operation for the computer to perform. In order for a program to function properly, each keyword must be used correctly and in the appropriate place. In addition to keywords, the Java language includes identifiers to allow programmers to name the Java classes, fields, and methods they create (Gilbert 29). Very specific, yet simple, rules exist for creating Java identifiers:

- An identifier must begin with an upper- or lowercase letter. While it is permissible to begin with an underscore (_) or a dollar sign ($), this is considered bad form by human readers.
- The digits (0-9) can be used as part of the identifiers, although they cannot be used to begin one.
- Identifiers are not restricted in length, but those longer than 20 characters quickly become tedious to type.
- The case of the letters used in identifiers is significant to Java; programmers must take care to use lowercase and uppercase characters accurately. (Gilbert 29)

Delimiters serve as the punctuation marks of the Java language. Like the commas, periods, and brackets in natural written language, delimiters group things, making clear where one ends and another begins. When a computer compiles a program written in Java, it first scans the source code looking for delimiters, then uses these delimiters to discover the structure of the program. A mistyped delimiter will thoroughly confuse the compiler, preventing the program from compiling and/or running.

Consequently, it is crucial that delimiters are typed correctly (Gilbert 29-30). The following characters are delimiters in the Java language: braces ({}), parentheses (()), semicolons (;), single quotation marks ('), double quotation marks ("), and colons (:).

Although each computer programming language will have its own specific rules that govern its use, some general guidelines for program development have evolved:

- Do not allow large programs to become monolithic; instead divide them into relatively independent, easy-to-understand pieces, known as the *code subroutine*, *procedure,* or *function* (Gilbert 19).
- Develop *pseudo code* to outline the essentials of a computer program using English statements and programming language-like key words and structures ("Computer" 1).
- Indent to show the structural relationship among the statements and to enhance the readability of the program ("Computer" 2).

In many instances, these guidelines are quite similar to those invoked in the use of natural language.

Comparison of Natural Language and Artificial Languages

The preceding brief history of computer programming languages demonstrates that these artificial languages are becoming increasingly similar to natural language. Specifically, a comparison of natural language and Java reveals many parallels between the two, including: (1) dedicated vocabulary, (2) importance of word order/placement, (3) grammatical rules that ensure accurate communication between author (i.e., programmer/writer) and audience (i.e., computer/reader), (4) chunking (i.e., subroutines/paragraphs), and (5) structure (as identified by delimiters and punctuation marks). In addition to these concrete elements shared by natural and artificial languages, theoretical similarities have been identified by researchers such as Mann (2003), Bresko (1991), and Corbin, Moell, and Boyd (2002).

Nancy Mann, drawing on the work of Randolph Quirk and Geoffrey Nunberg, states that punctuation rules derive from the "intrinsically public nature of writing in which the originator of the message is not usually present to clear up any difficulty in interpretation . . . [and] such difficulties are inevitable because . . . a writer can't control or even predict the circumstances of message reception" (363). Punctuation, then, is a "norm system" for "remote-controlling reader interpretation"; as such it must be highly conventionalized to ensure clear comprehension by all readers. Although punctuation symmetries are not perfect, they are valuable, providing the necessary function of information management – specifically, telling readers how to interpret relationships between and within propositions. Mann further contends that

> the analogy with computerized information processing isn't merely decorative; we have, after all, created computers in our own image. Punctuation decision rules rely on binary contrasts, a familiar cognitive mechanism, and they repeatedly apply a single criterion (fixed vs. unfixed location) to different kinds of entities (single words vs. whole statements), another familiar mechanism. They respect what appears to be a biological processing limitation of the human brain, the well-known fact that we can

> apprehend simultaneously, in parallel, only an average of seven items. . . . Like communication software, punctuation seems to work by signaling in advance the mental posture that the receiving party needs to assume in order to process upcoming information. Finally, like most computers, the punctuation system abhors ambiguity. (Mann 363-364)

Likewise, Laura Bresko notes similarities between computer programming and technical writing:

> Many similarities exist between computer programming and technical writing. For example, in the composition process for both programs and documents, programmers and technical writers gather all available, pertinent materials and begin writing in a logical order according to the rules of the language they are using. The difference is in the language and the audience. Programmers write programming code for the computer to interpret and writers write words and sentences for their readers to interpret. (218)

Michelle Corbin, Pat Moell, and Mike Boyd, commenting on Bresko's analogy, observe that her comparison "could be extended to assert that technical editing – and content editing in particular – provides the same quality assurance processes for technical information that software testing does for programming code" (287).

Computer Science (CS) and Computer Science Information Systems (CSIS) Majors in the Writing Classroom

The significant similarities between writing and programming as well as those between natural and artificial languages have been capitalized upon by numerous instructors (e.g., Hyler 1985; Pesante 1991; Levine, Pesante, and Dunkle 1991; Taylor and Paine 1993; and Kay 1998) who have introduced successful, productive initiatives in their classrooms to link computer programming and writing. In 1985, Linda Hyler, while teaching at a junior high school, married programming and writing by engaging her students in an innovative project that had them transfer a creative writing story activity to the computer using BASIC. Consequently, writing occurred at two levels: the creative writing of the story and the technical writing of the program (2). While such early attempts to introduce the computer to the English writing classroom obviously required tremendous ingenuity, less than 20 years later, computers have become a fixture in many – if not most – writing classrooms. In contrast to Hyler and her ilk, educators in the 21st Century face the inverse challenge: bringing writing into the Computer Science classroom.

Educators such as Harriet Taylor and Katherine Paine have met this challenge head-on with their inventive approach to teaching composition skills to CS/CSIS students in the technical writing classroom. Their approach includes relating the skills of technical writing to the skills students have already developed in their computer science courses; specifically, Taylor and Paine liken the writing process to that of software development (277). Students participating in Taylor's and Paine's technical writing classes are assigned a term paper in the area of programming languages with explicit specifications to research professional journals, develop formal proposals describing their research, develop outlines, and write papers using the main division of the outline as the relevant subdivisions of the papers. One of the significant outcomes of this assignment was that students became aware of the parallels between writing projects and software development projects (275).

Similarly, Linda Pesante, based on her belief that students can use their knowledge of software development to enhance their understanding of the writing process, invokes the power of analogy in her writing classroom. Specifically, she has found it effective to "draw parallels between the development of a software system and that of a document," and she cites examples of software engineering students talking about "engineering their papers" and attempting to "get it right during the design phase" (206). To overcome student resistance to writing, Pesante points out how some of their basic engineering skills (e.g., analytic and organizational skills) can help them understand the writing process (207).

Another writing instructor, David Kay, reiterates the value of using analogies to teach technical writing skills to CS majors. Kay's institution (UC Irvine) requires all students to take one upper-level course that concentrates on writing. One such course, offered by the UC Irvine Computer Science Department, focuses on communications issues relevant to computer scientists and computer professionals. To make writing more relevant to the CS students in this course, instructors take advantage of analogies between writing and software development in order to motivate CS students to pay more careful attention to their writing (117).

Levine, Pesante, and Dunkle also report the benefits of using analogies in the classroom. Specifically, they noticed a significant difference in their students' attitudes toward reviews of their writing after drawing analogies between producing a draft and producing a prototype (119-120). From their experiences working with CS students, these instructors have concluded that learning by analogy provides students with a "powerful mechanism for applying the skills they already have as computer scientists and software engineers" (117). The following table (Figure 1) presents many of the analogies noted and/or invoked by educators in teaching writing to CS/CSIS majors.

Figure 1: Writing/Programming Analogies

Writing process	Software development [6]
Linear model of writing process	Waterfall phase model[7]
Development of a document	Development of a software system[8]
Planning/Prewriting	Engineering[9]
Outlining/drafting	Design phase[10]; writing pseudo code[11]
Generating text	Writing code[12]
Grammar and syntax to facility audience understanding	Grammar and syntax to facility computer's comprehension[13]
Audience analysis	User analysis[14]
Reuse	Rapid prototyping[15,16,17]
Chunking	Subroutines[18]

[6] Taylor and Paine 277; Levine, Pesante, and Dunkle 117
[7] Levine, Pesante, and Dunkle 117
[8] Pesante 206
[9] Pesante 206
[10] Pesante 206
[11] Student comments
[12] Pesante 207
[13] Kay 119
[14] Pesante 207
[15] Pesante 207
[16] Levine, Pesante, and Dunkle 119-120
[17] Scacchi 7

The ability of instructors to utilize analogies between writing and computer programming to teach composition skills to Computer Science majors demonstrates that useful and meaningful similarities exist between computer programming and writing. Some of these similarities are listed below.

- *Logic and Organization:* Both programming and writing involve a logical, carefully organized exposition of complex ideas. Consequently, CS/CSIS students, with an understanding of the top-down design and stepwise refinement of software, should be easily able to approach prose the same way (Kay 119). Furthermore, their basic software engineering skills, such as their analytic and organizational skills, will help them as writers (Pesante 207).

- *Grammar:* Both writers and programmers must follow a set of language rules (Kay 119).

- *Iterations:* It is difficult, arguably impossible, to produce a document or a software system that is exactly what it should be on the first attempt. Rather, writing and programming are both iterative processes, subject to successive refinements. Planning, drafting, evaluating, and revising processes do not occur once and only once but many times and not necessarily in the same order each time. In writing the computer code to meet the needs of the user, the requirements of the system, and the constraints under which it must operate, software developers work iteratively, refining and enhancing the system at each iteration. In the same way, writers develop documents by

[18] Levine, Pesante, and Dunkle 119

going through drafts or *iterations*. Each iteration of the document is a successive approximation that comes closer to the final product, the purposeful document suitable for the intended audience (Kay 120; Pesante 207; Levine, Pesante, and Dunkle 119).

- *Approach (Prewriting):* Both writers and programmers, prior to actually writing or programming, must perform an audience/user analysis and a problem analysis to clearly define their task in view of the goals and constraints of the project. Effort put into a task at this crucial planning stage pays large dividends in the process since refinements made at this point are much less disruptive to the project than refinements that occur at the end. Specifically, problems in software development are easier and less expensive to solve during the early stages of development because it is easier to modify a design than it is to fix a large, complex software system. Similarly, writers must make deliberate choices about what to attend to in each iteration of a document in order that they solve higher-level problems before lower-level problems (Pesante 207; Taylor and Paine 275; Levine, Pesante, and Dunkle 119).
- *Review:* Evaluation is important both in the process of writing and in the process of software development. Writers and programmers must check their developing products against the requirements, evaluate their progress against the original plan, and engage in peer reviews/user testing before finalizing

their products in order to ensure they will satisfy audience/end user needs and expectations (Pesante 207).

- *Linear Model:* The *waterfall* phase model, a linear model of the software development process, demonstrates how software development proceeds through an orderly and linear sequence from one phase to the next. Although this model does not accurately reflect the iterative process of developing large complex software systems, it helps software designers structure their work and plan effectively. Similarly, linear models of the writing process show an orderly sequence of phases: (1) prewriting, (2) writing, and (3) revision. Even though the writing process is generally *not* linear, the phases of these models assist writers in structuring or chunking work on a writing project (Levine, Pesante, and Dunkle 117).

- *Chunking:* The notion of chunking (i.e., dividing a writing project into logical parts that can be tackled independently) in the writing process is remarkably similar to the use of subroutines in computer programming, whereby a large program is divided into smaller "mini-programs" that can later be combined efficiently into a single, functioning larger software program. CS/CSIS students can apply the principles behind the familiar software development strategy of creating subroutines to better manage their writing tasks (Levine, Pesante, and Dunkle 119).

- *Reuse*: Reuse is the concept of recycling portions of a computer software program or an existing document for inclusion in a developing program or document. However, in order to benefit from reuse, systems or documents must be composed of modules or components (e.g., subroutines or chapters) that can be easily inserted into the new system/document. For example, in the writing process, the writer may reuse or adapt the format, organizational structure, or rhetorical moves of a model document into an entirely new document (Levine, Pesante, and Dunkle 119-120).
- *Prototyping*: Prototyping is the creation of a "reduced functionality version of a software system early in its development" that allows many software design activities to be skipped or glossed over (Scacchi 7). The prototypes are then used to test proof of concept and get early feedback from users, thereby identifying problems and addressing them early on. Similarly, writers construct a draft – a reduced but functional version of the document – that peers can evaluate. Revisions and refinements are then made to the draft. Another significant commonality between prototypes and drafts is that neither is thrown away; rather each evolves into subsequent drafts and, ultimately, the final product (Levine, Pesante, and Dunkle 119-120).

Most of the observations detailed by the researchers above focus on the specific analogies and techniques educators use in the technical writing classroom to make connections between the writing process and the software development process for

CS/CSIS majors. The present study seeks to determine whether (1) CS/CSIS students are aware of the existing overlap in the cognitive structures of computer programming and writing and (2) whether these students believe the overlapping structures will assist them in more readily mastering the composition skills and strategies presented in the technical writing classroom.

Methodology and Procedures

Setting

The study was conducted in various technical writing classrooms at Kennesaw State University (KSU) in Kennesaw, Georgia. The university, a large suburban institution with a student population of approximately 18,000, is dedicated to remaining on the cutting edge of technology; therefore, all sections of technical writing are taught in computer classrooms with state-of-the-art projection equipment. Additionally, to ensure students receive appropriate guidance and attention from the instructor, class size is limited to 25 students per section.

The Computer Science and Information Systems Department at KSU requires all Computer Science (CS) and Computer Science Information Systems (CSIS) undergraduate majors to take the technical writing course offered by the KSU English Department; consequently, the overwhelming majority of students in these classes are CS/CSIS majors. Although the CS and CSIS programs have a different emphasis, both undergraduate programs are based on a strong technical foundation that includes programming principles, systems analysis, systems architecture, data communications, and database design and management. The CS degree program requires more upper-level mathematics; the CSIS degree program requires additional business courses and includes more business applications.

Undergraduate CS/CSIS students receive an education that is practical, technical, analytical, quantitative, conceptual, and current. Upon graduation, these students are expected to be fluent in Windows, UNIX, Oracle, C++, Java, Visual Basic, and many other technologies. Consequently, graduates typically accept positions that include systems analysis, programming, data communications, end-user support, database administration, consulting, and top management (Kennesaw State University [KSU]).

Specifically, CS majors at KSU receive a blend of the foundations of computer science and applications in the information technology (IT) industry, with an emphasis on the study of computer systems architecture, software development, and data communications. Core technology areas include programming, computer architecture, operating systems, data communication, systems analysis and design, database applications, and project management. These core areas are supported by a strong foundation in computing principles, such as the design of programming languages, data structures, and operating system principles. The program includes a significant mathematics component, and mathematics concepts are incorporated into many of the major courses (KSU). Appendix A provides the degree requirements for an undergraduate majoring in Computer Science; Appendix B is a sample program of study for a full-time undergraduate student majoring in Computer Science.

Undergraduates in KSU's CSIS program receive a solid foundation in IT principles and practice. Emphasis is on IT applications rather than on the computer itself. Core technology areas include programming, computer architecture, operating systems, data communication, systems analysis and design, database applications, and project management. The program of study also includes practical statistics, IT organizations, financial systems, and a significant general business component that is integrated into many CSIS courses (KSU). Appendix C provides the degree requirements for an undergraduate majoring in Computer Science Information Systems; Appendix D is a sample program of study for a full-time undergraduate student majoring in Computer Science Information Systems.

Based on the program of study required of CS/CSIS majors at KSU, these students, by the time they enroll in the technical writing course (usually in their junior or senior year), typically possess the requisite knowledge of computers and computer programming necessary to provide insightful and valuable responses to the questions posed in both the surveys and interviews relative to this study. Consequently, CS/CSIS majors enrolled in technical writing at KSU proved an excellent pool of subjects.

Participants

Lincoln and Guba note that, when selecting participants, "the sample is to be selected in ways that will provide the broadest range of information possible" (102). This notion has been followed in the selection of participants for this study. Although this study is limited to a particular subset of university students (i.e., CS/CSIS majors), attempts were made to accommodate "the broadest range" by purposefully seeking out a setting in which the greatest diversity of CS/CSIS majors would be represented. Specifically, a writing course that all CS/CSIS majors are required to take was selected, resulting in a population of subjects that mimics the diversity of the population of majors within the Computer Science and Systems Information Department overall.

All subjects who participated in this study were enrolled in a section of English 3140, Technical Writing, at Kennesaw State University during the Fall 2003 or Spring

2004 semester. Each section met once a week for one hour and fifteen minutes over a 15-week semester; students were required to engage in class activities online in lieu of a second classroom meeting. During the Fall 2003 semester, four sections of technical writing were offered at KSU. Appendix E provides a sample syllabus from one of these sections. Because of scheduling conflicts, only one of those sections was available for participation in this study. Of the 25 potential subjects from that section, only 13 completed both the pre- and post-test questionnaires. Consequently, the pool of subjects was significantly smaller than had been originally anticipated. To compensate for the small number of participants, the researcher elected to gather additional data in the Spring 2004 semester. Two sections of the technical writing course taught during Spring 2004 participated in this study at the discretion of the technical writing instructor for each section. Of a potential 50 subjects, 33 completed both pre- and post-tests (the remaining 17 were lost to attrition or declined to participate in the research study), bringing the total number of subjects for this study to 47.

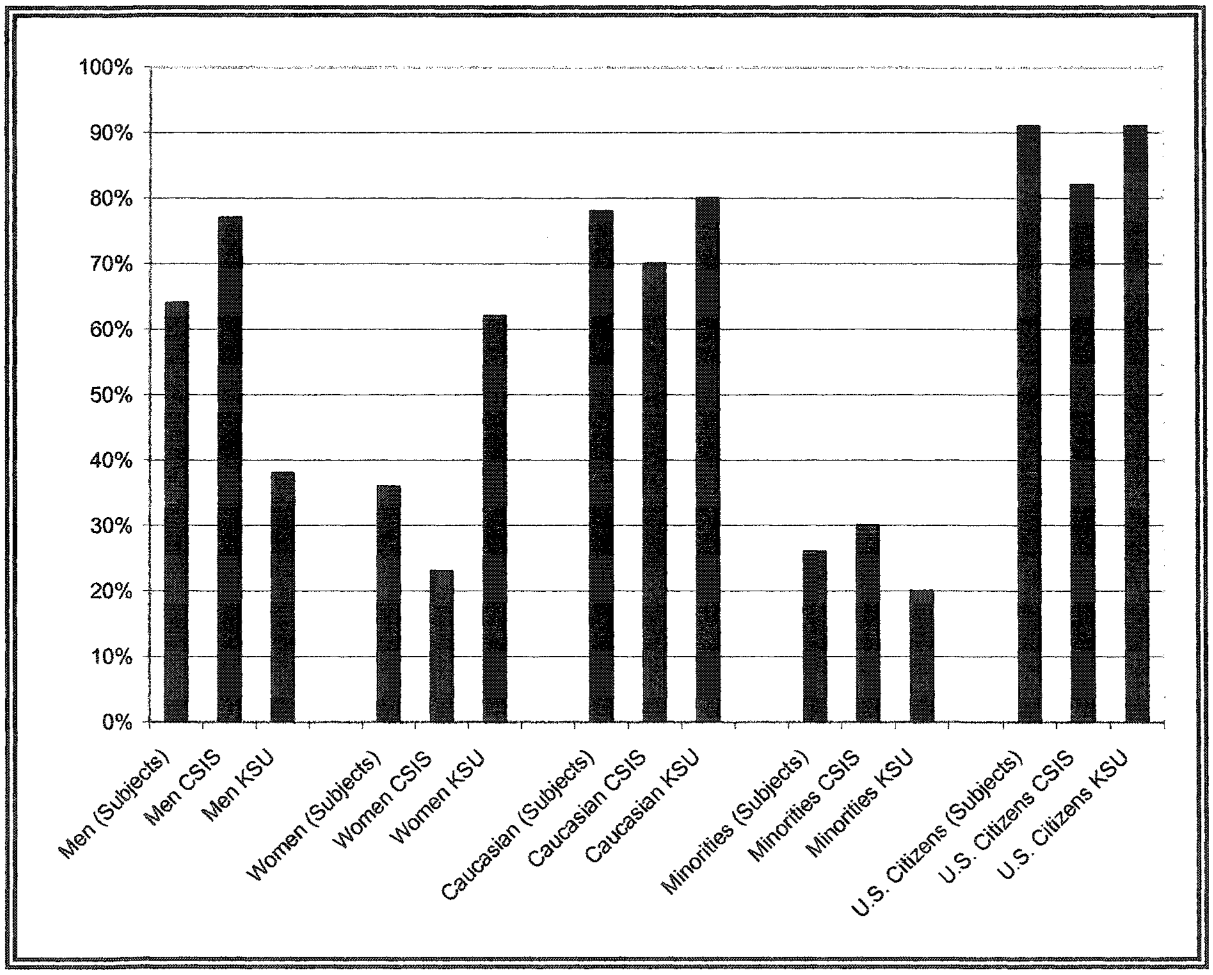
Although the technical writing course is open to all KSU students, the Computer Science and Systems Information Department requires all of its majors to take this course. Consequently, CS/CSIS majors typically constitute the overwhelming majority of students in these courses – which proved true for the subjects of this study, 96% of whom were CS/CSIS majors. Given the large proportion of CS/CSIS majors among the total subjects, it was anticipated that the demographics of the study participants would reflect that of CS/CSIS majors overall. However, this expectation was only partially met in terms of gender (64% of the subjects were male, 36% female), whereas ethnicity (78%

Caucasian, 26% minorities) and citizenship (91% U.S. citizens, 9% non-U.S. citizens) were more in line with the demographics of the general KSU population.

The most recent data on the student population at KSU (Fall 2004) indicates that women outnumber men by a ratio of nearly 2:1 (62% women, 38% men), an overwhelming majority of the student body is Caucasian (80%), and 91% are U.S. citizens. Meanwhile, within the Computer Science and Information Systems Department, men outnumber women by a ratio of nearly 3:1 (78% men, 22% women) – the inverse of the general student population. Caucasians constitute the majority of CS/CSIS students (70%) but to a lesser degree than in the general student population (KSU, "Fact Book"; KSU "Computer Science and Information Systems Department"). Figure 2 provides a graphical representation of a demographics comparison among study participants, CS/CSIS majors, and the general KSU student population.

Other relevant demographics data were also retrieved in the study. Specifically, 19% of the participants spoke English as their second language, most were upperclassmen (17% juniors; 72% seniors), and all had completed prerequisite English courses. Only one student had taken an additional writing course (in Communications). Of the 47 subjects, 36% were currently employed in their major or had prior work experience in their field, and 25 subjects (53%) stated that they currently or in a previous position were required to perform on-the-job writing tasks. Five subjects (11%) had more than 10 years of experience writing work-related documents, three (6%) had 6-10 years of experience, ten (21%) had 3-5 years of experience, and 7 (15%) had less than two years of experience writing on the job. When asked to assess their own writing abilities, 4% of the subjects believed their writing ability to be **excellent**, 53% rated their ability as **good**, 32% thought their writing was only **adequate**, and 11% described their writing as **fair**.

Figure 2: Demographics Comparison



A self-assessment of computer programming skills provided somewhat different results: 19% of the subjects believed their computer programming skills to be **excellent**, 19% rated their skills as **good**, 38% thought their programming skills were only **adequate**, 13% described their skills as **fair**, and 11% claimed their computer programming skills were **poor**. When asked to cite the computer programming languages in which they are fluent, 22 subjects (47%) listed Java, 16 (34%) HTML, 16 (34%) C++, and 8 (17%) Visual Basic. Other languages cited by only one or two students include Perl, Cobol, ALGOL, and ASP. In a follow-up question, subjects were to indicate those computer programming languages with which they were familiar but not fluent. In response, 13 subjects (28%) cited Java, 17 (36%) HTML, 14 (30%) C++, and 18 (38%) Visual Basic. Other languages with which students expressed some familiarity include Perl (5), Cobol (7), ALGOL (1), FORTRAN (6), Pascal (6), Ada (5), LISP (3), and PROLOG (2).

Instruments

Data was collected in the form of pre- and post-test questionnaires and via interviews. At the start of the semester, subjects completed (1) a demographics questionnaire (Appendix F) and (2) a pre-test self-reporting questionnaire (Appendix G) designed to collect information regarding subjects' perceptions of their writing skills, their expectations regarding the technical writing course, and their perceived connections between programming and writing and between natural and artificial languages.

Near the end of the semester, subjects completed a post-test self-reporting questionnaire (Appendix H). The questions in this document paralleled those in the pre-test questionnaire in order to determine whether subjects' attitudes and perceptions changed during the course of the semester. Additionally, five subjects were selected at random to participate in follow-up interviews that allowed them to elaborate on the topics addressed in their post-test questionnaires.

Data Collection

The pre- and post-test questionnaires, administered as part of this research study, produced sufficient quantitative data to allow for a general overview of participants' attitudes and perceptions regarding commonalities between computer programming and writing and between artificial and natural languages. However, the findings from this research with perhaps the greatest practical applicability in the writing classroom were derived from follow-up questions and interviews, which offered tremendous insight into students' perspectives.

This study was largely qualitative because the researcher ascribes to the belief system of naturalistic inquiry and believed that the research questions could best be answered by collecting data in this manner. According to Lincoln and Guba, five beliefs are central to naturalistic inquiry; all five were adhered to in this study:

- Axiom 1 – *The nature of reality*: realities are multiple, constructed, and holistic
- Axiom 2 – *The relationship of knower to the known*: knower and known are interactive, inseparable
- Axiom 3 – *The possibility of generalization*: only time and context-bound working hypotheses are possible

- Axiom 4 – *The possibility of causal linkages*: all entities are in a state of mutual simultaneous shaping so that it is impossible to distinguish causes from effects
- Axiom 5 – *The role of values in inquiry*: inquiry is value-bound

(37-38)

Additionally, Lincoln and Guba describe fourteen "characteristics of operational naturalistic inquiry" that "display a synergism such that, once one is selected, the others more or less follow." The fourteen characteristics, along with a short description of how they were applied in this study, are as follows (39-43).

1. *Natural Setting*: The phenomena of the study take their meanings as much from their contexts as they do from themselves. Consequently, care was taken not to influence students' perceptions of the topics being investigated in order to obtain the truest data from their responses, as emerged within the natural context of the classroom environment.
2. *Human Instrument*: The human instrument has the characteristics necessary to cope with an indeterminate situation. In this study, the instructor and the students contributed to the overall making of meaning.
3. *Utilization of Tacit Knowledge*: Tacit knowledge is experiential knowledge or commonsense, which is intuitive and felt. Such knowledge was useful in phrasing follow-up questions and interview questions in order to elicit meaningful responses from study participants.

4. *Qualitative Methods*: Lincoln and Guba contend that the "human-as-instrument is inclined toward methods that are extensions of normal human activities: looking, listening, speaking, reading, and the like. . . . [T]he human will tend, therefore, toward interviewing, observing, mining available documents and records, taking account of nonverbal cues, and interpreting inadvertent unobtrusive measures" (199). Consequently, follow-up questions, interviews, group discussions, and observation were used to gather data to answer the research questions of this study.
5. *Purposive Sampling*: Based on informational rather than statistical considerations, purposive sampling is done for a specific purpose – to maximize information, not facilitate generalization. Such sampling depends on the flow of information as the study is conducted rather than on prior considerations. Given this, the data for this study was collected exclusively from students taking the KSU English 3140 Technical Writing course in order to assure a subject pool that would provide the most valuable data in response to the research questions posed in this study.
6. *Inductive Data Analysis*: This analysis is the process of making sense of field data (interviews, observations, documents, unobtrusive measures, nonverbal cues, and other information pools). For the purpose of this study, data were analyzed on an ongoing basis in order to note the evolution of the study.
7. *Grounded Theory*: Grounded theory is the theory that emerges from the data as opposed to have an *a priori* theory drive the data collection. Such a process

handles multiple realities and makes transferability dependent on local contextual factors. The data collected in this research study were used to develop a theory that responds to the questions that drive this study and to shape new ones in order to explain what was observed.

8. *Emergent Design*: Emergent design is the notion that the study must "emerge (flow, cascade, unfold)" rather than be constructed *a priori* "because it is inconceivable that enough could be known ahead of time about the many multiple realities to devise the design adequately" (41). Therefore, it was anticipated that this study would evolve and grow – changing as quickly as needed to more completely describe what was being observed. In the case of qualitative research, change is a good thing.
9. *Negotiated Outcomes*: The participants of the study were permitted to see and comment on the findings. It is their realities that this research sought to reconstruct, and the quality of the findings depended on the interaction between the knower and the known.
10. *Case Study Reporting Mode*: The case study reporting mode is ideal for providing thick description. This mode is highly responsive to Lincoln and Guba's five axioms of the naturalistic paradigm, and it is an ideal vehicle for communicating with the subject. Although case studies were initially planned as part of this research study, they were deemed counterproductive (i.e., they could potentially have tainted post-test data) during the Fall 2003 semester and were not conducted in either the Fall or the Spring semester.

11. *Idiographic Interpretation*: Idiographic interpretation states that what is found in a particular context has meaning only for that context during that particular time. This concept is particularly meaningful for this research investigation since each subject brought to the study a unique knowledge and experience base that informed his or her mastery of writing. Consequently, the data were interpreted specifically within the context in which they were gathered and during the particular time of the study.
12. *Tentative Application*: The concept of tentative application is that the findings of a particular study cannot be generalized to other contexts although information supplied from one study may make possible a judgment of transferability to another similar context. In drawing conclusions from the data gathered in this study, care was taken to avoid making broad generalizations and applications of the findings.
13. *Focus-Determined Boundaries*: The term "focus-determined boundaries" refers to the inquiry boundaries set "on the basis of the emergent focus . . . because that permits the multiple realities to define the focus (rather than inquirer preconceptions)" (Lincoln and Guba 42). In keeping with the establishment of such boundaries, this study was limited to the perceptions and perspectives of CS/CSIS majors participating in a technical writing course. Specifically, the study sought to determine if the subjects noted any similarities between computer programming and writing in order to facilitate their mastery of technical writing.

14. *Special Criteria for Trustworthiness*: Lincoln and Guba suggest credibility in place of internal validity, transferability in place of external validity, dependability in place of reliability, and conformability in place of objectivity to achieve trustworthiness of data and the findings. To achieve such trustworthiness, the researcher engaged in persistent observation, triangulation, and member checking in conducting this study.

Data Sources, Collection Strategies, and Triangulation

To answer the questions posed in this study and to further describe those that emerged during the course of the research, three data collection methods were used. These were selected in order to maintain multiple sources of evidence and to increase the strength and the credibility of the study (Lincoln & Guba 1985, Erlandson et al. 1993, Stake 1995). These methods include: (1) questionnaires (demographics, pre-test, and post-test), (2) follow-up questions, and (3) interviews.

Questionnaires

The questionnaires used in conducting this study were designed to elicit both quantitative and qualitative data through the use of, respectively, a Likert scale and open-ended questions. Though qualitative methods of research are typically associated with the naturalistic-constructivist type of study described here, Erlandson et al. note that "mainstream researchers regularly use qualitative methods, and naturalistic researchers will often use quantitative methods" (35). Consequently, the methods of data-gathering and the type of data gathered were deemed to be in keeping with the research paradigm used in this study. Additionally, Erlandson et al. argue that it is not whether both quantitative and qualitative measures are used to gather data nor the order of the measures used that distinguishes naturalistic studies from more conventional studies. Rather, what is crucial here is "whether the combined measures are designed to reduce or expand the constructions of reality that are being considered" (37).

Follow-Up Questions

Follow-up questions were developed for each subject selected as an interviewee. These questions were formulated based on the quantitative responses on the pre- and post-test questionnaires. The answers students provided to the follow-up questions helped shape the final interview questions and allowed for more in-depth dialogue between the researcher and the subjects.

Interviews

Interviews with students played a significant role in the data collection process for this research. Interviews, as noted by Lincoln and Guba, have the advantage of allowing respondents "to move back and forth in time – to reconstruct the past, interpret the present, and predict the future, all without leaving a comfortable armchair" (273). Interviews were conducted with five students selected at random; these interviews provided a means of triangulating the data gathered from questionnaires and follow-up questions.

Data Analysis

Since several methods of data collection were used in this study, several stages of analysis were necessary. Specifically, Miles and Huberman state that data analysis consists of three activities that flow together: "data reduction, data display, and conclusion drawing/verification" (10). These three stages were applied to the management of the data in this research investigation.

Stage One: Data Reduction

Data reduction is the process of selecting, focusing, simplifying, abstracting, and transforming the data gathered in the research study. According to Miles and Huberman, "data reduction is a form of analysis that sharpens, sorts, focuses, discards, and organizes data in such a way that final conclusions can be drawn and verified" (11); they further suggest that data can be reduced through selection and summary, and by being subsumed in a larger pattern. Their conclusions suggest that any research is best served by eliminating irrelevant information; consequently, irrelevant data were deleted during the analysis stage of this study, resulting in considerably fewer subjects than initially anticipated. Additionally, the data were organized into an electronic spreadsheet that facilitated a close and thorough examination.

Stage Two: Data Display

Data display refers to the ways in which data can be organized, compressed, and assembled in order to reach conclusions about the message that data has to convey. Miles and Huberman state that displays are "designed to assemble organized information into an immediately accessible, compact form so that the analyst can see what is happening and either draw justified conclusions or move on to the next step of analysis that the display suggests useful" (11). Further, they suggest that a display will place the data in a visual format so that the viewer can "draw valid conclusions and take needed action" (91).

In Stage Two, the data entered into the electronic spreadsheet were matrixed such that they could be visually organized to display multiple data sets for analysis. For example, graphs, tables, and charts were created to allow for an examination of the data from a variety of perspectives in order to make possible meaningful insight into emerging patterns and trends. Using a computer to capture and depict the data in different visual arrays proved to be the best means for identifying and defining emergent patterns in this study.

Stage Three: Conclusion Drawing and Verification

In stage three, a number of visual representations, produced to view the collected data, served as the springboard from which conclusions about findings were drawn. Miles and Huberman state that qualitative analysts decide what their observations mean from the first "chunk" of data that is collected; however, final conclusions are not drawn until the data collection process is complete.

Description and Discussion of the Data

The data gathered in this study are both quantitative and qualitative. The quantitative data can be broadly divided into three categories: demographic information (used to classify subjects and to assist in interpreting qualitative findings); self-assessment of computer programming abilities and writing abilities; and responses to pre- and post-test questionnaires designed using a Likert scale to elicit students' level of agreement or disagreement with statements related to writing, computer programming, and the relationship between the two. The first two categories were described above in the discussion regarding study participants; the responses to the pre- and post-test questionnaires are discussed below. The qualitative data is drawn from written responses to the questionnaires (wherein students were asked to elaborate on the responses indicated on the Likert scale), follow-up questions, and student interviews.

Likert Responses

The first part of the pre- and post-test questionnaires asked students to respond, using a Likert scale, to statements regarding their writing skills, computer programming skills, and their perceived relationship between the two. Figure 3 presents the averages of student responses to the pre- and post-tests; it also shows the degree of change in students' attitudes between the pre- and post-tests.

Students' level of agreement changed for each of the eight statements from pre- to post-testing. These differences range from 0.06 to 0.92. For three statements (7, 2, and 6 – listed in descending order), student agreement increased; for the remaining five statements (3, 1, 8, 4, and 5 – listed in descending order), the level of agreement decreased from pre- to post-testing. The slight change in students' attitudes from pre- to post-testing suggests study participants' perceptions were only mildly affected, if at all, by their participation in a semester-long writing course[19]. Although the slight change in students' attitudes does not seem particularly remarkable, some apparent contradictions in students' responses to these two questionnaires warrant a closer examination.

Figure 3: Pre- and Post-Test Comparison of Quantitative Data

	Pre/Post Test Responses	Pre-Test	Post-Test	*Difference**
1	Writing skills may/did improve.	8.43	7.96	(0.47)
2	Existing knowledge base may/did contribute to my ability to master the writing skills and strategies presented in this course.	7.19	7.87	0.68
3	My understanding of computer languages may/did assist me in mastering the strategies of effective technical and professional writing.	6.07	5.15	(0.92)
4	I believe many strategies for writing may be/are similar to the strategies used to write computer code.	5.6	5.3	(0.30)
5	I believe I might better understand the rules of grammar and the theories of composition if the instructor used analogies related to programming languages to make connections that relate to me.	4.8	4.51	(0.29)
6	I believe the rules of grammar may be similar to the rules that govern computer programming.	5.46	5.53	0.07
7	I believe there is little or no similarity between computer programming languages and the English language.	4.77	5.66	0.89
8	The writing process I used to write an essay may/did change because of the writing strategies and theories I will/did learn in this course.	7.74	7.38	(0.36)

**Note: parentheses indicate a decrease from pre- to post-testing.*

[19] A conscious effort was made by the researcher not to influence subjects' perceptions of any connection between programming and writing; consequently, no specific discussions regarding the potential similarities between the two were conducted prior to the post-test.

First, although the post-test questionnaire indicated an increase in students' perception that their existing knowledge base could/did contribute to their ability to master writing skills and strategies, it also showed a decrease in students' belief that their understanding of computer languages – arguably a component of their knowledge base – could facilitate their mastery of these same skills and strategies. Second, the end-of-the-semester post-test questionnaire reflected an increase (albeit slight) in students' perception that the grammar rules of computer programming languages may be similar to the grammar rules of the English language, yet students' belief that there is "little or no similarity between computer programming languages and the English language" increased and their perception that analogies related to programming languages might help them master writing decreased. Finally, although students indicated an increased belief in the similarity of grammar rules between computer programming languages and the English language, their post-test questionnaire responses revealed a decreased perception that strategies for coding computer programs were similar to those used in writing using natural language. These contradictions will be explored more fully in "Findings and Key Recommendations."

Written Comments and Interview Responses

The qualitative data was gathered from pre- and post-test questionnaires, follow-up questions, and interviews with students. An item-by-item analysis of participants' responses to the pre- and post-test questionnaire statements follows.

Statement 1, *My writing skills may/did improve,* elicited a fairly high level of agreement from subjects on the pre-test questionnaire. On the post-test questionnaire, study participants rated their overall writing improvement as less than they had anticipated on the pre-test, yet they were able to cite specific ways in which their writing had improved (e.g., "learned to gear my writing towards my audience better," "communicate more efficiently," and "grammar, punctuation"). This seeming discrepancy may perhaps be explained by their increased knowledge of correct writing, grammar, and punctuation yet still inexperienced ability to apply that knowledge in order to correct their own writing errors. As one student noted in his interview, "I learned my faults in my writing style but have yet to fix the problem." Alternatively, perhaps subjects realized that writing is not a "problem" that can be solved once and for all but rather a skill that can be continually improved upon, thereby making it more difficult to accurately quantify their degree of improvement.

Statement 2, *My existing knowledge base may/did contribute to my ability to master the writing skills and strategies presented in this course*, was purposely vague to determine whether the students perceived a knowledge of computer programming as a potential asset to them in a writing course. Although there was a slight increase in subjects' agreement with this statement from pre- to post-testing, none of the respondents explicitly cited their experience with computer programming as part of the existing knowledge base that might assist them. Comments ranged from broad and imprecise (e.g., "college education; past experiences" and "I think existing knowledge always contributes to further learning") to much more specific (e.g., "many years of experience in various business environments" and "a basic understanding of syntax"). Only one subject, perhaps based on his perception of what is entailed in a technical writing course, even remotely connected knowledge from his major field of study to writing: "My knowledge of IT systems will aid in my writing because its terminology will be use in the writing. My knowledge of operating systems and applications will help with my writing."

A more focused follow-up to the preceding item, Statement 3, *My understanding of computer languages may/did assist me in mastering the strategies of effective technical and professional writing*, was an attempt to direct subjects' responses toward an area particularly relevant to this study: a potential link between an understanding of computer programming languages and mastery of writing skills and strategies. Additionally, students were asked to list, and in later interviews elaborate on, the specific artificial languages and elements of those languages that they thought were most beneficial in helping them to master writing skills and strategies.

In the pre-test, subjects indicated that their attitudes toward this statement were fairly neutral, albeit with a slight leaning toward agreement, and knowledge of the programming language Java was most frequently cited (11 occurrences) as likely be an asset to students in a technical writing class. C++ was cited by eight respondents, and other languages mentioned by only one or two subjects include COBOL, HTML, Eiffel, SQL, Perl, Visual Basic, and ASP. Some respondents elaborated on their selections, claiming that "maybe Java computer language will help because it is organized and well indented" and that "COBOL is a procedural language and very structured in its syntax."

Others offered more general observations that suggested an agreement with the notion that computing languages can be an asset in understanding writing strategies, such as "Writing programs requires following a logical path which can be applied in technical writing"; still others stressed their lack of agreement with this notion, noting that "concepts such as transition and document flow are not naturally associated with programming."

In the post-test questionnaire, Statement 3 witnessed the most dramatic change (a 9% shift) of the eight statements on the questionnaires. Subjects' attitudes from pre- to post-testing remained essentially neutral, albeit now with a slight leaning toward disagreement, and Java remained the most cited language as having the potential to assist students in understanding writing concepts. SQL and C++ were each mentioned one time; all others were no longer cited. Again, some subjects offered explanations for their choices. One supported his claim that Java is useful because of its "regular logical semantics structure," and another stated that SQL has a "syntax [that] helps one write better."

In response to follow-up questions and in their interviews, respondents offered their perspectives on how/why they believe (or do not believe) knowledge of computer programming can assist CS/CSIS students in mastering writing skills. While some of these responses were rather broad generalizations, such as "step-by-step approach to both coding and writing," "logically structured," and "none; don't think that it really relates," others offered tremendous insight into students' perspectives and perceptions:

> There is no question that to be a good programmer you should be a good writer because detail is very important. However, I don't believe that the reverse is true.
>
> I think the process I have gathered as a computer science major has assisted me (not a particular language). I am more aware of the logical order in which I should write to help others understand a new process. Programming requires specific rules to be followed or it won't compile. English is the same way.
>
> Understanding that you write the code so that the computer understands and is useful. In English, you have to communicate in a manner that the audience understands.

Statement 4, *I believe many strategies for writing may be/are similar to the strategies used to write computer code,* was intentionally vague to allow respondents flexibility and creativity in interpreting the statement and responding to it. Although this tactic elicited confusion from a few subjects (e.g., "Do not understand" and "Not sure"), the majority of the subjects offered surprisingly consistent responses.

In the pre-test questionnaire, the Likert scale revealed that respondents were overall neutral on this topic, with a slight tendency toward disagreement. Yet, in the comments portion of the questionnaire, nearly two-thirds of them were able to cite specific strategies or concepts that are similar in writing and computer programming. (This apparent contradiction will be addressed more completely in "Findings and Key Recommendations.") These similarities can be divided broadly into three major categories, identified here in composition terms as (1) prewriting, (2) organization, and (3) syntax. The post-test showed similar findings, with a slightly increased tendency toward disagreement (as indicated by the Likert scale); the written comments on the post-test, the follow-up questions, and the interviews again fell into the three broad categories identified above, although in different proportions. In the pre-test, five subjects indicated that elements of prewriting were common to both computer programming and composition; in the post-test/interviews, twelve respondents saw such a connection. Organization, on the other hand, remained a constant factor, with five subjects citing it in the pre-test and five in the post-test/interviews. The final item cited by more than one respondent, syntax, was mentioned by two respondents in the pre-test and two in the post-test. Figure 4 presents a sampling of student responses from these three categories.

Other students chose to introduce terms from their knowledge of computer programming to create analogies (e.g., "Developing *classes* reminds me of outlining a paper" and "Outlining is similar to writing *pseudo-code*"), while still others noted that they saw no specific similarities at all.

On the pre-test questionnaire, Likert scale responses to Statement 5, *I believe I might better understand the rules of grammar and the theories of composition if the instructor used analogies related to programming languages to make connections that relate to me,* revealed a slight tendency toward disagreement; the post-test indicated a slight increase in this disagreement. In the comments portion of the questionnaires and in follow-up questions and interviews, subjects were to provide examples of programming analogies that would help them better understand composition. Most respondents had difficulty identifying such analogies; only five offered specific analogies:

> Relational modeling (UML).
> Operators & syntactical units: clauses, subject-predicate, prepositions, etc.
> There are grammar rules in programming that would help.
> Describe grammar logically as adhering to a highly structured set of specifications.
> Re-emphasize that it all follows a rule and/or pattern.

Figure 4: Student Responses to Statement 4

Categories	Subject Responses
Prewriting	Break it down; what are the steps.
	Audience considerations and breaking a problem into smaller pieces are shared components of both.
	You have to plan out your code before you write it, the same as you need to plan your paper.
	When writing code, you start with planning before coding. In writing, you write an outline before writing a paper.
	The processes are similar, like brainstorming, outlining, etc.
	The main similarity is the existence of the planning process and its impact on the final product.
	You must understand how the computer/audience must get the data.
	You have to define what you want out of each before you start.
	You need to plan what will be said in both cases before you start.
Organization	Just as you would outline a paper in order to gather your thoughts, one should take the same approach with coding.
	Writing programs requires following a logical path which can be applied in technical writing.
	Proper order.
	Logical organization.
	Mainly in organization (e.g., programs have to be written with a certain flow in mind). I have found in this course it helps to create an outline before writing.
	The order in which a written document is put together follows certain guidelines, much like the layout of a program.
	The logical flow of statements in both (grammar).
	Strategies to write include how to set up a paragraph, and coding requires certain flow for a program.
Syntax	Syntax = grammar rules.
	They have a certain structure and punctuation.
	Use of syntax.
	You have to have the right syntax or the code will not work.

The remaining respondents, for the most part, were uncertain of how to respond, saw no analogies between the two, or doubted that such analogies would be an asset. Others were cautiously optimistic (e.g., "I don't see how this would help, but I'm open to trying anything") or identified potential pitfalls of such a practice (e.g., "I think the non-technical majors wouldn't understand"). One subject even claimed that the analogies should work in the opposite direction: "Programming teachers should use analogies relating to composition."

Responses to Statement 6, *I believe the rules of grammar may be similar to the rules that govern computer programming*, changed slightly toward agreement (0.07) between the pre- and post-tests but essentially reveal a stable neutral response from the study participants. Despite a lack of strong agreement with this statement, subjects were able to cite several examples of computer programming rules that are similar to the rules of English grammar. The most frequently cited example (nearly 20 comments) was syntax:

> Punctuation, structure.
> If you don't write the code according to syntax, it is useless. The same concept applies to English.
> When writing code, punctuation is critical for execution in the program.
> Grammar has rules that must be followed which is similar to syntax when you have to write in a specific way that the computer understands and can read.
> You have to be clear and concise in writing in either case. There are very specific rules for both.
> Each has specific syntax and specific use to receive the required outcome.
> Commas and semicolons.

However, a variety of other similarities – and in some cases differences – were noted as well:

> Programming syntax is much stricter.
> Higher level programming languages are more similar to grammar than computerational (sic) techniques.
> There are fewer exceptions in programming. For example, compare "ie" rules to the use of the ";" in C++. Also, I think this comparison may confuse some since the punctuation often exists in both.

> The English language ideally should be able to be parsed into syntactic units just as compilers parse code.
> Classes => nouns; operations => verbs.
> In grammar you must have a subject and a verb; likewise, in programming you must have a method and a procedure.

For Statement 7, *I believe there is little or no similarity between computer programming languages and the English language*, there was a noticeable change in subjects' attitudes between the pre- and post-tests – nearly a full point toward agreement, bringing the general sentiment to a more neutral perception of this issue. Students were also asked to provide arguments or explanations for why they agreed or disagreed with Statement 7. Figure 5 presents respondents' comments divided into categories of No Similarities, Neutral, and Similarities.

Statement 8, *The writing process I used to write an essay may/did change because of the writing strategies and theories I will/did learn in this course*, though similar to Statement 1, is meant to focus more on the overall process of writing rather than on specific writing skills. Though there was a slight decrease in agreement from the pre- to post-test questionnaire, the overall perception seems to be that respondents expected their writing process to change during their participation in the technical writing course and that this expectation was duly met.

As a follow-up question to Statement 8 and a topic of discussion in interviews, students were asked to indicate the specific ways their writing process had changed. Though a few respondents indicated that their writing process had not changed, the majority commented that they had altered the prewriting stage of their composing process as a result of the technical writing course – particularly, in terms of writing for a specific audience and planning/outlining their documents:

> I gear my papers more to the appropriate audience.
> Know your audience.
> I now plan before I start and I consider who I am writing for.
> Better planning before starting the writing process; use of outline to help in the planning process.

Figure 5: Students' Comments in Support of Statement 7 Responses

No Similarities	English is not objective enough.
	English is more free flowing.
	I do not see many similarities. Programming is very logical and has strict rules. English has too many exceptions to the rule to be similar to programming languages.
	English is too complex; much ambiguity.
	English is a bit more difficult to master.
	It is a different language by definition. If we could code in English we would be doing it. Computer languages are not at all like the English language.
	English is too vague to be strongly related to a programming language.
	I believe Boolean logic is most commonly used in programming, and the true/false nature of this logic is very different from the shades of gray one finds in English.
	Writing English is totally different than programming with a computer language. I do not have to know grammar, it [coding] is more fun, and it is challenging.
	Each computer language has its own structure, just as English does.
Neutral	Depends on what type of writing one might attempt.
	It depends on what programming language one is using. Higher-level languages are closer to plain English than lower-level languages.
	Punctuation is used very differently between the two. However, I also disagree because there is a definite protocol to both.
Similarities	Programming statements tend to follow English for clarity and ease of use.
	They are both ways to communicate, just with different audiences.
	We write code based on grammar principles.
	Write from left to right.
	You have to learn it in the manner it is taught. Code is useless without syntax, as is English.
	Many programming languages and codes are very understandable because it is the English language.

The responses to the questionnaires, both revealing and enlightening, raised two additional questions that were presented to subjects during interviews:

1. How has your approach to composition been affected by your knowledge and understanding of computer languages and computer programming, if at all?
2. How has your approach to computer programming been affected, if at all, by the writing strategies, grammatical rules, and composing process presented in this technical writing course?

In response to the first question, interviewee comments were quite diverse. One respondent stated that he believed his computer programming skills provided him the ability to "handle large ideas as fairly self-contained components." Specifically, he was able to relate the practice of writing numerous subroutines for a large program to the similar task of collaborative writing wherein members of a group or team each write a section of a larger document by first breaking it into more manageable, logical pieces. Another subject stated that she applied her practice of writing pseudo-code before programming to her writing assignments by writing outlines prior to drafting a document – something she was previously not in the habit of doing. Other respondents noted that they carried over the concepts of organization, structure, grammatical rules, and patterns from the study of computers and applied these concepts to their writing assignments.

Regarding the question of whether the skills and strategies learned in the technical writing course had any impact on their approach to computer programming, one of the subjects stated that he did not believe his writing skills could contribute to his ability to program; another thought programming and writing complemented one another but was unable to verbalize any specifics. Of the three remaining interviewees, one noted that the study of writing made her "logically think about where [she] should place punctuation –

which is important in programming because if you forget a comma or appropriate punctuation, the program won't run." Another interview participant found paragraph development to be "similar to writing functions" when programming. Consequently, he believed that his knowledge of paragraphing improved his ability to write functions. The final interviewee was only able to state in broad terms that the knowledge he acquired in the technical writing course helped him "to understand items in a sequence and with a specific syntax."

Findings and Key Recommendations

The purpose of this study was to determine (1) whether Computer Science (CS) and Computer Science Information Systems (CSIS) majors recognize the overlapping cognitive structures that exist between computer programming and English composition and (2) whether these students believe that the overlapping cognitive structures assist them in mastering the writing skills and strategies presented in a technical writing course. However, the study itself was permitted to drive the direction of the research, and the scope was accordingly expanded to include an examination of numerous other similarities between programming and writing in order to facilitate the research questions that evolved from the preliminary findings.

In preparing and conducting this study, the researcher had certain expectations regarding research participants that were not realized. Specifically, the researcher expected (1) that study participants would see analogies between writing and computer programming (e.g., process, grammar, and language similarities) and (2) that such analogies would matter to them. Study participants' failure to meet these expectations opened a previously unforeseen avenue for exploration into the value of CS/CSIS analogies in the technical writing classroom. Furthermore, the researcher also had anticipated that study participants' perceptions, as revealed through the pre- and post-test questionnaires, would show a more substantial shift than this study revealed – presumably in ways that affirmed students' recognition of and appreciation for similarities in the study of composition and their studies in their major disciplines. Although none of these expectations were realized, the findings of this study may be, arguably, much richer, more diverse, and more useful than had originally been anticipated.

Findings

This research study produced unexpected, and at times contradictory, results. These results are analyzed in detail below.

1. Although the post-test questionnaire indicated an increase in students' perception that their existing knowledge base could/did contribute to their ability to master writing skills and strategies, it also showed a decrease in students' belief that their understanding of computer languages – arguably a component of their knowledge base – could facilitate their mastery of these same skills and strategies. This shift in students' attitudes suggests that (1) they recognize existing knowledge can be called upon and applied to new academic challenges, yet (2) they cannot identify any computer programming skills or strategies that are transferable to writing. This finding demonstrates that while similarities, analogies, and overlapping cognitive structures between writing and computer programming are obvious to many instructors (e.g., Hyler 1985; Pesante 1991; Levine, Pesante, and Dunkle 1991; Taylor and Paine 1993; and Kay 1998), they are not necessarily apparent to the CS/CSIS students studying writing.

2. The end-of-the-semester post-test questionnaire reflected study participants' increased belief in a similarity between the grammar rules of English and those of computer programming languages, yet participants professed a decreasing belief that the English language and computer programming languages are similar and indicated a decreasing perception that computer programming analogies might help them master writing. That is, they recognize *grammatical* similarities between natural and artificial languages, but they are unable to see the overall likenesses between the two types of languages, nor how those likenesses might suggest transference of concepts (and/or the application of cognitive structures). Thus, participants see similarities at a lower-order level of reasoning (e.g., in applying syntax rules) but are unable to recognize parallels at higher-order levels of reasoning (e.g., at conceptual levels). Although these results are at odds with one another, they have significant implications for the classroom, particularly in regards to students' learning styles and the design of the writing course.
3. Though participants recognized a similarity between the syntax of computer programming languages and that of the English language, post-test questionnaire responses revealed a decreased student perception that strategies for programming were similar to those used in writing. As in Finding 2 above, students recognize *grammatical* similarities between natural and artificial languages, but they profess an inability to identify similarities in the *processes* these different types of languages use. This finding, again, suggests

that while CS/CSIS students attend to lower-order level of reasoning similarities between natural and artificial languages they are less adept at recognizing overlaps at a higher-order level of reasoning. That is, they do not recognize the conceptual, abstract commonalities that exist between the processes of coding and writing. The implications for a technical writing course are discussed in Key Recommendations.

4. Despite the observations and findings of academic scholars and researchers that indicate numerous commonalities between natural language and computer programming languages and between the writing process and the software development process, the majority of these similarities are not readily and consciously apparent to the CS/CSIS students who participated in the current study. Furthermore, students' participation, during the course of the current study, in a technical writing class that did not include *explicit mention of such similarities* apparently did little to increase their ability to recognize the commonalities that have been identified by researchers.
5. Although researchers have identified analogies between natural and artificial languages and between writing and computer programming and although some researchers have successful used these analogies to teach writing skills and strategies to CS/CSIS students, most of the CS/CSIS students in the current study did not believe such analogies would be useful to them in mastering writing. Despite the fact that they perceived analogies to be of little value, a few study participants were able to identify analogies, often the same

ones that had been identified by instructors and researchers. These findings indicate that CS/CSIS students are able to make connections (i.e., recognize analogies) between writing and coding regardless of whether they find any value in acknowledging those connections; furthermore, the findings suggest that perhaps students are applying analogies of which they are not consciously aware. That is, it may be hypothesized that the rules and strategies that govern computer programming and artificial languages have become so deeply ingrained in the knowledge base of CS/CSIS students that the similarities between programming and writing and between natural and artificial languages prompt students to draw upon the same cognitive structures in the use of each, resulting in an unconscious transference of the general rules and strategies of coding to the practice of writing.

6. Computer programming languages are becoming increasingly similar to natural language. Specifically, the earliest artificial languages, which closely approximated mathematical notations (e.g., FORTRAN), have evolved into much more sophisticated programming languages that bear a greater resemblance to natural language than they do to the machine language which gave rise to them. For example, the Java computer programming language uses delimiters – such as parentheses, semicolons, and quotation marks – to group the source code and delineate the structure of a program in much the same way that punctuation marks are used to provide structure and guidance in written texts. Additionally, Java possesses a vocabulary that includes

keywords (i.e., reserved words that indicate the operation the computer is to perform) and identifiers (i.e., names created by programmers to designate the classes, fields, and methods they create) that are much more similar to natural language than they are to the binary numbers of the earliest machine language, which bore no resemblance to the written word. As the disparity between computer programming languages and natural language diminishes, writing in natural language and in artificial languages may increasingly invoke the use of the same cognitive structures. In consequence, student proficiency in computer programming may facilitate an ability to learn college-level writing skills. That is, CS/CSIS students' understanding of higher-level programming languages that closely approximate natural language may provide cognitive structures upon which students can draw to facilitate their mastery of writing and upon which instructors teaching writing to CS/CSIS majors can draw for analogies to relate writing to computer programming. Conversely, a strong understanding of natural language and how to use it correctly and efficiently in the writing process may become increasingly useful to CS/CSIS majors in their understanding and mastery of computer programming languages.

Key Recommendations

1. Writing instructors should not assume CS/CSIS students can recognize and apply the similarities, analogies, and cognitive structures that exist between writing and computer programming. Rather, instructors must explicitly

identify analogies between the writing process and the software development process and between natural and artificial languages. More importantly, these instructors must aid students in applying their existing cognitive structures (i.e., knowledge of artificial languages and programming) to the challenges they face in the writing classroom. For example, instructors might begin the semester by providing equivalent verbiage for similar concepts and tasks that exist between the disciplines of English Composition and Computer Science, such as *writing process/software development, linear model/waterfall phase model, planning/engineering,* and *outlining/writing pseudo-code* (see Figure 1: Writing/Programming Analogies). Next, the instructor might demonstrate that similar processes and elements exist in both disciplines, such as the need for audience/user analysis, the use of grammars, repeated iterations, and the like (see the subsection titled "Computer Science Majors in the Writing Classroom" for a more extensive list of similarities). Finally, the instructor might develop activities and assignments that encourage students to recognize similarities between writing and programming. For example, students might compare a paragraph of text with a subroutine in a computer program to identify similarities such as punctuation, content, structure, purpose, and so on. Future research should focus on emerging technologies in the area of Computer Studies to identify ways in which students' knowledge of these technologies can be used by writing instructors to facilitate the ability of their CS/CSIS students to master writing strategies and techniques.

2. Study participants' ability to see similarities between natural and artificial languages in the application of syntax rules demonstrates their ability to attend to technical details and to comprehend details at a lower-order level of reasoning. Similarly, their inability to recognize parallels between natural and artificial languages at broader, conceptual levels suggests an inability to apply higher-order levels of reasoning in this instance. These results suggest that CS/CSIS majors have learning styles and abilities that facilitate their ability to recognize technical similarities and parallels at a lower-order level of reasoning more easily than those at a higher-order level of reasoning. If this interpretation is indeed true, it raises the question of whether other learning style characteristics are shared by many of the CS/CSIS majors. An understanding of the learning-style similarities specific to this particular student population could significantly affect the way in which their instructors design their courses. In a class in which the majority of the students are in the same academic discipline and tend to share a similar learning style, courses could be structured to build upon these students' learning-style strengths and to find ways to overcome their learning-style weaknesses. For example, given CS/CSIS majors' understanding of the overlap in syntax rules between natural and artificial languages, a writing course designed specifically for these students might employ analogies that draw upon students' understanding of the syntax of computer languages to explain the syntax of natural language. Moreover, CS/CSIS majors' understanding and use of flowcharts and their

familiarity with the concept of parsing as it relates to computer programming languages suggest that sentence diagramming may be an effective learning device. Such an analytic approach to language may best aid these students in understanding writing strategies since they have honed their analytic skills in their major area of study. Notably, these concepts have greater applicability than just for CS/CSIS majors in the writing classroom. Anytime a body of students share a particular learning style and/or way of knowing, instructors of any subject would be well advised to discern what those learning styles are and how to take advantage of the strengths and compensate for the weaknesses inherent in that particular style of learning.

3. Students' ability to recognize *grammatical* similarities between natural and artificial languages but their inability to identify similarities in the *processes* these different types of languages use suggests that CS/CSIS students are skilled in attending to lower-order-level-of-reasoning similarities between natural and artificial languages but less adept at recognizing overlaps at higher-order levels of reasoning. Specifically, they do not recognize the conceptual, abstract commonalities between the processes of coding and writing, which suggests that CS/CSIS majors may have learning styles and skills that facilitate some abilities and skills over others. Such findings indicate that an instructor's understanding of his or her students' learning styles could significantly affect course design – in a CS/CSIS writing class or any class in which students share a particular learning style.

4. Students' inability to recognize the commonalities between natural language and computer programming languages and between the writing process and the software development process, which have been identified by researchers, demonstrates that such similarities are not readily and obviously apparent to students. Furthermore, without explicit mention of these similarities, simultaneous involvement in programming assignments and writing projects does not enhance students' skill in recognizing the commonalities that exist. Consequently, writing instructors who are aware of such commonalities and hope to use them to foster students' writing success must explicitly identify such similarities, demonstrate how the similarities indicate the use of overlapping cognitive structures, and guide students in utilizing these similarities and cognitive structures in ways that foster good writing. For example, instructors might draw specific analogies between writing and coding (e.g., use of syntax, formatting, etc.), then demonstrate how the two processes can be analogized by examining and drawing parallels between the various steps in each process (e.g., prewriting/engineering, drafting/writing pseudo-code, etc.). While the findings and recommendations of this study are specific to CS/CSIS majors in the writing classroom, the general principle has transferability to practically any discipline that can identify within itself analogies to another field. Of course, practical applicability demands a fairly standard knowledge base among the group of students being taught – which is, admittedly, often not the norm, especially in general education courses.

5. CS/CSIS students' ability to recognize analogies between natural and artificial languages and between writing and computer programming (although they profess not to see the value of these) and researchers'/instructors' successful use of these analogies to teach writing skills and strategies to CS/CSIS students suggest that analogies may be powerful tools in facilitating CS/CSIS students' mastery of writing. If indeed both writing and computer programming draw upon the same cognitive structures, explicit analogies may not only aid students in making connections between two disciplines and better understanding a new academic challenge, they could also potentially reinforce the common cognitive structures, resulting in improved performance in both disciplines. This use of interdisciplinary analogies is applicable beyond the areas of CS/CSIS and English; in fact, such analogies could be implemented in any discipline that is able to identify commonalities between itself and other disciplines common to its students. Of course, the value of analogies is maximized when the majority of the students share a common knowledge base, one often developed in the pursuit of a shared field of study. Consequently, future research should explore (1) similarities between disciplines that could be the foundation for analogies, (2) the use of learning communities (i.e., assigning a specific group of students to common sections of particular courses) to facilitate the use of analogies in the learning process, and (3) the effect the use of interdisciplinary analogies has on student learning.

6. The increasing similarities between computer programming languages and natural language add fuel to the argument that programming and writing rely on the same cognitive structures. While the current study has focused on the contribution an understanding of computer programming languages can make to the mastery of written English, the diminishing disparity between programming languages and natural language suggests that an important reciprocity may exist. That is, while knowledge of programming may facilitate the ability to write well, perhaps writing well also facilitates students' programming skills. Future research should explore the potential reciprocity between these two disciplines and identify a means of exploiting this reciprocity to enhance student learning. Finally, broadening the scope of future studies to explore reciprocal learning in other seemingly disparate disciplines that perhaps share cognitive structures could lead to improved course design, more effective teaching, and enhanced student learning.

While this study has focused primarily on the overlapping cognitive structures between computer programming and writing, the findings have implications beyond the areas of Computer Science, Computer Science Information Systems, and English Composition. Research on overlapping interdisciplinary cognitive structures for any given subset (i.e., major) of the student population at the university level may lead to a greater understanding of how instructors might more effectively teach specific segments of university students in courses that lie both within and outside of their major area of study.

Bibliography

Baumann, James F., and Ann M. Duffy-Hester. "Making Sense of Classroom Worlds: Methodology in Teacher Research." *Methods of Literacy Research: The Methodology Chapters from the Handbook of Reading Research, Vol. III.* Ed. Kamil, Michael L. Mahwah, NJ: Lawrence Erlbaum Associates, Inc., 2002. 1-22. 18 Dec. 2003 <http://emedia.netlibrary.com/reader/ebook_info.asp?cu_id=63465>.

Bell-Gredler, Margaret. *Learning and Instruction: Theory into Practice.* NY: Collier Macmillan, 1986.

Ben-Ari, Mordecahi. "Constructivism in Computer Science Education." *SIGCSE Bulletin* 30.2 (1998): 257-261.

Bickerstaff, Douglas D., and Judith D. Kaufman. *Improving Student Writing Skills: Inter-Departmental Collaborations.* Proc. of the Twenty-Third ACM Special Interest Group on Computer Science Education (SIGCSE) Technical Symposium on Computer Science, 1992, Kansas City, Missouri. NY: ACM Press, 1992. 42-45. <http://portal.acm.org>.

Biggs, John B. "Introduction and Overview." *Teaching for Learning: The View from Cognitive Psychology.* Ed. John B. Biggs. Australia: Acer, 1991. 1-6.

Biggs, John B., and Phillip J. Moore. *The Process of Learning*, 3rd edition. New York: Prentice Hall, 1993.

Birnbaum, June Cannell. "Reflective Thought: The Connection between Reading and Writing." *Convergences: Transactions in Reading and Writing.* Ed. Bruce T. Petersen. Urbana, IL: NCTE, 1986. 30-45.

Bissex, Glenda L., and Richard H. Bullock, eds. *Seeing for Ourselves: Case-Study Research by Teachers of Writing.* Portsmouth, NH: Heinemann, 1987.

Bresko, Laura L. "The Need for Technical Communicators on the Software Development Team." *Technical Communications* (1991): 214-220.

Chandler, Kelly. "Working in Her Own Context: A Case Study of One Teacher Researcher." *Language Arts* 77.1 (1999): 27-33. *ProQuest.* GALILEO. Kennesaw State University, Kennesaw, GA. 29 Dec. 2003.

"Computer Programming." *AccessScience* (2002). *McGraw-Hill.* GALILEO. Kennesaw State University, Kennesaw, GA. 02 Jul. 2003.

Collins, Allan M., and M. Ross Quillian. "Experiments on Semantic Memory and Language Comprehension." *Cognition in Learning and Memory.* Ed. Lee W. Gregg. NY: John Wiley & Sons, Inc., 1990. 117-137.

Cook, Vivian. "Knowledge of Writing." *International Review of Applied Linguistics in Language Teaching* 39.1 (2001). *EBSCOhost.* GALILEO. Kennesaw State University, Kennesaw, GA. 01 Aug. 2003.

Corbin, Michelle, Pat Moell, and Mike Boyd. "Technical Editing as Quality Assurance: Adding Value to Content." *Technical Communication* 49.3 (2002): 286-300.

Donnelly, Colleen Elaine. *Linguistics for Writers.* Albany, NY: State University of New York Press, 1994.

Dorsel, Thomas N. “Conflicting Goals: A Dilemma for the Teacher-Researcher.” *Teaching of Psychology* 8.1 (1981): 52-53.

Ellis, Donald G. *From Language to Communication.* Mahwah, NJ: Lawrence Erlbaum Associates, Inc., 1999.

Erlandson, David A., Edward L. Harris, Barbara L. Skipper, and Steve D. Allen. *Doing Naturalistic Inquiry: A Guide to Methods.* Newbury Park, CA: Sage Publications, 1993.

Fell, Harriet J., Viera K. Proulx, and John Casey. *Writing Across the Computer Science Curriculum.* Proc. of the Twenty-Seventh ACM Special Interest Group on Computer Science Education (SIGCSE) Technical Symposium on Computer Science Education, 1996, Philadelphia, Pennsylvania. NY: ACM Press, 1996. 204-209. <http://portal.acm.org>.

Flower, Linda, and John R. Hayes. “The Cognition of Discovery: Defining a Rhetorical Problem.” *The Writing Teacher’s Sourcebook.* Ed. Gary Tate and Edward P. J. Corbett. New York: Oxford University Press, 1988. 92-102.

Flower, Linda, and John R. Hayes. “A Cognitive Process Theory of Writing.” *College Composition and Communication* 32.4 (1981): 365-387.

Gagné, Robert Mills. “The Acquisition of Knowledge.” *Psychological Review* 69.4 (1962): 355-365.

Gagné, Robert Mills, and Marcy Perkins-Driscoll. *Essentials of Learning for Instruction.* 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1988.

Gilbert, Stephen. *Sams Teach Yourself Java 2 Online in Web Time.* Indianapolis, IN: Sams Publishing, 1999.

Hammack, Floyd M. "Ethical Issues in Teacher Research." *Teachers College Record* 99 (1997): 247-265.

Harley, Trevor A. *The Psychology of Language: From Data to Theory.* East Sussex, UK: Psychology Press Ltd., 1995.

Hartman, Janet. *Writing to Learn and Communicate in a Data Structures Course.* Proc. of the Twentieth ACM Special Interest Group on Computer Science Education (SIGCSE) Technical Symposium on Computer Science Education, 1989, Louisville, Kentucky. NY: ACM Press, 1989. 32-36. <http://portal.acm.org>.

Hyler, Linda. "Teaching Writing Through Programming." *Computers and Composition: An International Journal for Teachers of Writing* 2.2 (1985): 2-3. <http://www.hu.mtu.edu/~candc/archives/v2/2_2_2_Hyler.html>.

Kaczmarczyk, Lisa C. *A Technical Writing Class for Computer Science Majors: Measuring Student Perceptions of Learning.* Proc. of the Thirty-Fourth ACM Special Interest Group on Computer Science Education (SIGCSE) Technical Symposium on Computer Science Education, 2003, Reno, Nevada. NY: ACM Press, 2003. 341-345. <http://portal.acm.org>.

Kay, David G. "Computer Scientists Can Teach Writing: An Upper Division Course for Computer Science Majors.: *SIGCSE Bulletin* 30.2 (1998): 117-120.

Keenan, Edward L. "Some Properties of *Natural Language* Quantifiers: Generalized Quantifier Theory." *Linguistics and Philosophy* 25 (2002): 627-654.

Kennesaw State University (KSU). Computer Science and Information Systems (CSIS) Department. *Computer Science & Information Systems.* 2003. 19 Dec. 2003. <http://science.kennesaw.edu/csis/>.

Kennesaw State University (KSU). Fact Book. 2004. 25 May 2004. <http://ir.kennesaw.edu/fb>.

Knoblauch, C. H. "The Teaching and Practice of 'Professional Writing.'" *Writing in the Business Professions.* Ed. Myra Krogen. Urbana, IL: NCTE, 1989. 246-264.

Kreymer, Oleg. "An Evaluation of Help Mechanisms in Natural Language Information Retrieval Systems." *Online Information Review* 26.1 (2002). *ProQuest.* GALILEO. Kennesaw State University, Kennesaw, GA. 23 Jul. 2003.

Levine, Howard, and Howard Rheingold. *The Cognitive Connection: Thought and Language in Man and Machine.* New York: Prentice Hall, 1987.

Levine, Linda, Linda H. Pesante, and Susan B. Dunkle. "Implementing the Writing Plan: Heuristics from Software Development." *The Technical Writing Teacher* 18.2 (1991): 116-126.

Lincoln, Yvonna S., and Egon G. Guba. *Naturalistic Inquiry.* Beverly Hills, CA: Sage Publications, 1985.

Lycan, William G. *Logical Form in Natural Language.* Cambridge, MA: MIT Press, 1986.

Mann, Nancy. "Point Counterpoint: Teaching Punctuation as Information Management." *College Composition and Communication* 54.3 (2003): 359-393.

McClendon, Ronald C. "Construct Validity of the MSLQ and Cognition as Related to Motivation and Learning Strategy Use of Preservice Teachers." Diss. U. of Akron, 1993.

Miles, Matthew B., and A. M. Huberman. *Qualitative Data Analysis: An Expanded Sourcebook.* 2nd ed. Thousand Oaks, CA: Sage Publications, Inc., 1994.

Pesante, Linda H. *Integrating Writing into Computer Science Courses.* Proc. of the Twenty-Second ACM special Interest Group on Computer Science Education (SIGCSE) Technical Symposium on Computer Science Education, 1991 San Antonio, Texas. NY: ACM Press, 1991. 205-209. <http://portal.acm.org>.

Pfeiffer, Phil. "What Employers Want from Students: A Report from OOPSLA." *SIGCSE Bulletin* 31.2 (1999): 69-70.

Phillips, D. C., and Jonas F. Soltis. *Perspectives on Learning.* NY: Teachers College Press, 1985.

"Programming Languages." *AccessScience* (2002). McGraw-Hill. GALILEO. Kennesaw State University, Kennesaw, GA. 01 Aug. 2003.

Reagan, Sally Barr. "Teaching Reading in the Writing Classroom." *Journal of Teaching Writing* 5 (1986): 177-185.

Rubin, Herbert J., and Irene S. Rubin. *Qualitative Interviewing: The Art of Hearing Data.* Thousand Oaks, CA: Sage Publications, 1995.

Scacchi, Walt. Models of Software Evolution: Life Cycle and Process. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1987.

Smith, Frank. *Understanding Reading: A Psycholinguistic Analysis of Reading and Learning to Read.* 5th ed. Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers, 1994.

Stake, Robert E. *The Art of Case Study Research.* Thousand Oaks, CA: Sage Publications, Inc., 1995.

Sternglass, Marilyn S. "Integrating Instruction in Reading, Writing+, and Reasoning." *The Writer's Mind.* Ed. Janice N. Hays et al. Urbana, IL: NCTE, 1983. 153-158.

Sternglass, Marilyn S. "Writing Based on Reading." *Convergences: Transactions in Reading and Writing.* Ed. Bruce T. Petersen. Urbana, IL: NCTE, 1986. 151-152.

Taylor, Harriet, and Katherine M. Paine. *An Inter-Disciplinary Approach to the Development of Writing Skills in Computer Science Students.* Proc. of the Twenty-Fourth ACM Special Interest Group on Computer Science Education (SIGCSE) Technical Symposium on Computer Science Education, 1993, Indianapolis, Indiana. NY: ACM Press, 1993. 274-278. <http://portal.acm.org>.

Tierney, Robert J., and Margie Leys. "What is the Value of Connecting Reading and Writing?" *Convergences: Transactions in Reading and Writing.* Ed. Bruce T. Petersen. Urbana, IL: NCTE, 1986. 15-29.

Travers, Robert M. W. *Essentials of Learning: An Overview for Students of Education,* 2nd ed. NY: The Macmillan Company, 1967.

Walker, Henry M. "Writing Within the Computer Science Curriculum." *SIGCSE Bulletin* 30.2 (1998): 24-25.

Wittrock, M. C. "The Generative Process of Memory." *The Human Brain*. Ed. M. C. Wittrock. Englewood Cliffs, NJ: Prentice-Hall, Inc, 1977. 153-184.

APPENDIX A

Computer Science Undergraduate Major Degree Requirements

Degree Requirements 2003-2004

Course	Description		Hours	Course	Description	Hours	
ANTH 2105 *or*	Anthropology	2	2	ART 1107 or	Visual Arts	3	3
GEOG 2105 *or*	Geography		2	MUSI 1107 or	Music	3	
PSYC 2105 *or*	Psychology		2	THTR 1107	Theater	3	
SOCI 2105	Sociology						
				ECON 1100 or	Global Econ	3	3
COM 1109 *or*	Communication	3	3	ECON 2100	Microeconomics	3	
FL 1002 *or*	Foreign Language		3				
PHIL 2200	Ways of Knowing		3	ENGL 1101	Composition I	3	9
				ENGL 1102	Composition II	3	
PHYS 1111	Intro. Physics I	8	4	ENGL 2110	World Literature	3	
&							
PHYS 1112	Intro Physics II		4	MATH 1113	Precalculus	3	7
o				MATH 1190	Calculus I	4	
PHYS 2211	Princ. Physics I		3				
PHYS 2211L	Lab		1	HIST 1110	World Civilization	3	9
PHYS 2221	Princ. Physics II		3	HIST 2112	America 1890	3	
PHYS 2212L	LAB		1	POLS 1101	Am. Government	3	
(PHYS 2211/2212 recommended)				HPS 1000	Fitness for Living	3	3

Lower Division Major Requirements			PREREQUISITES
MATH 2202	Calculus II	4	MATH 1190
CSIS 2300	Principles of Computing	3	NONE
CSIS 2301	Programming Principles I	3	CSIS 2300 & any credit-lvl MATH
CSIS 2302	Programming Principles II	3	CSIS 2301
CSIS 2520	Introduction to Data Communications	3	CSIS 2301

Upper Division Major Requirements			PREREQUISITES
CSIS 3150	Programming Languages	3	CSIS 2302
CSIS 3310	Database Design and Management	3	CSIS 2301
CSIS 3401	Introduction to Data Structures	3	CSIS 2302
CSIS 3402	Advanced Data Structures & Algorithms	3	CSIS 3401
CSIS 3510	Organization and Architecture	3	CSIS 2302
CSIS 3530	Operating Systems	3	CSIS 2520 & CSIS 3510
CSIS 3600	Systems Analysis and Design	3	CSIS 3310
CSIS 4500	Data Communications Protocols	3	CSIS 2520 & CSIS 3510
MATH 3322	Discrete Modeling I	3	MATH 1113 or MATH 2590
MATH 3332	Probability and Statistics	3	MATH 1190
MATH 4322	Discrete Modeling II	3	MATH 3322 & CSIS 2301
or			
MATH 3260	Linear Algebra	3	MATH 1190
COM 3385	Organizational Presentation	3	COM 1109 or COM 1129
or			
ENGL 3140	Professional Writing in the Disciplines	3	ENGL 2110
PHYS 3340	Digital and Analog Electronics	4	PHYS 1112 or PHYS 2212

Major Electives - Degree Program Tracks (Choice of one track)

Systems Track (12 hours)		**PREREQUISITES**
CSIS 4130	Parallel and Distributed Architecture and Algorithms	CSIS 3510 & CSIS 3150
CSIS 4560	Distributed Object Technology	CSIS 4500
CSIS 4730	Real-Time Systems and Simulation	CSIS 3530
CSIS 4850	Senior Project	CSIS 3600

Object-Oriented Software Development Track (12 hours)		**PREREQUISITES**
CSIS 3650	Object-Oriented Software Development	CSIS 3600
CSIS 4620	Object-Oriented Methods	CSIS 3650
CSIS 4650	Advanced Object-Oriented Software Development	CSIS 3650
CSIS 4850	Senior Project	CSIS 3600

Free Electives (Any courses in KSU curriculum totaling 8 hours)

Hours Required for Graduation

General Education	47
Lower Division Major Requirements	16
Upper Division Major Requirements	40
Major Electives	12
Free Electives	+ 8
TOTAL HOURS	**123**

- This form should only be used as a condensed summary. Please refer to the 2003-2004 KSU undergraduate catalog for complete details and official graduation requirements
- CS majors should enroll in major requirements as soon as possible. Do not complete your general education requirements before starting your CSIS coursework!
- Students wishing to graduate under this catalog must earn a grade of C or better in all major requirements (both upper and lower-division) and major electives
- Co-operative Education (CSIS 3396) and Internship (CSIS 3398) can only be used to fulfill free elective requirements

03-04 CS REQ. Last modified July 7, 2003.

APPENDIX B

Computer Science Undergraduate Major Sample Program of Study

2003-2004 Computer Science Sample Program of Study
Four-Year Program of Study for Full-Time Student

Status	1st Semester Program		2nd Semester Program	
Freshman (up to 30 hrs)	ENGL 1101	(3)	ENGL 1102	(3)
	MATH 1113	(3)	MATH 1190	(4)
	CSIS 2300	(3)	POLS 1101	(3)
	HPS 1000	(3)	COM 1109 or FL 1002 or PHIL 2200	(3)
	ECON 1100 or ECON 2100	(3)	CSIS 2301	(3)
	TOTAL HOURS:	**15**	**TOTAL HOURS:**	**16**
Sophomore (30 - 60 hrs)	ENGL 2110	(3)	HIST 1110	(3)
	CSIS 2520	(3)	MATH 3322	(3)
	PHYS 1111 or PHYS 2211	(4)	CSIS 3310	(3)
	MATH 2202	(4)	CSIS 3401	(3)
	TOTAL HOURS:	**17**	PHYS 1112or PHYS 2212	(4)
			TOTAL HOURS:	**16**
Junior (60 - 90 hrs)	CSIS 3402	(3)	MATH 3332	(3)
	CSIS 3510	(3)	MATH 4322 or 3260	(3)
	ART 1107 or MUSI 1107 or THTR 1107	(3)	CSIS 3530	(3)
	HIST 2112	(3)	CSIS 3150	(3)
	ANTH 2105 or GEOG 2105 or PSYC 2105 or SOCI 2105	(2)	Free Elective*	(2)
	TOTAL HOURS:	**14**	**TOTAL HOURS:**	**14**
Senior (over 90 hrs)	CSIS 3600	(3)	CSIS 4850**	(3)
	CSIS 4500	(3)	COM 3385 or ENGL 3140	(3)
	PHYS 3340	(4)	Free Elective*	(3)
	Track Elective**	(3)	Free Elective*	(3)
	Track Elective**	(3)	Track Elective**	(3)
	TOTAL HOURS:	**16**	**TOTAL HOURS:**	**15**

* Prerequisites for electives vary by class. Check KSU catalog for current prerequisite requirements.
** Systems Track: CSIS 4130, CSIS 4560, CSIS 4730
** O-O Software Development Track: CSIS 3650, CSIS 4620, CSIS 4650

This proposed schedule is not intended to reflect the availability of courses, but instead, is intended to demonstrate the feasibility of completing the B.S. Computer Science by a full-time student in a standard four-year schedule. Each student should consult "Schedule of Credit Courses" for an accurate listing of course offerings. Part-time students should consider enrollment in the Summer Semester in order to stay on schedule for graduation. All students should plan their schedule around the major required courses, and begin their CSIS coursework as early as possible in their program of study.

Students should note that the CSIS department recommends that they begin taking CSIS coursework as soon as possible and save their free electives until the end of their program of study.

Students should consult the current KSU Undergraduate Catalog for official degree requirements.

03-04 CS SAMPLE STUDY. Last modified July 11, 2003.

APPENDIX C

Computer Science Information Systems Undergraduate Major Degree Requirements

Information Systems
Degree Requirements 2003-2004

Course	Description	Hours	
ANTH 2105 or	Anthropology	2	2
GEOG 2105 or	Geography	2	
PSYC 2105 or	Psychology	2	
SOCI 2105	Sociology	2	
COM 1109 or	Communication	3	3
FL 1002 or	Foreign Language	3	
PHIL 2200	Ways of Knowing	3	
SCI 1101 &	Inter. Science Basic Princ.	4	7-8
SCI 1102	Inter. Science Issues in Science	3	
(The following classes may be substituted for SCI 1101/1102)			
CHEM 1111/1111L or		4	
CHEM 1211/1211L or		4	
PHYS 1111/1111L or		4	
PHYS 1112/1112L		4	

Course	Description	Hours	
ART 1107 or	Visual Arts	3	3
MUSI 1107 or	Music	3	
THTR 1107	Theater	3	
ECON 1100 or	Global Econ	3	3
ECON 2100	Microeconomics	3	
ENGL 1101	Composition I	3	9 hrs
ENGL 1102	Composition II	3	
ENGL 2110	World Literature	3	
MATH 1101 &	Math Modeling	3	6
MATH 1106	Elem App Calc	3	
HIST 1110	World Civilization	3	9
HIST 2112	America 1890	3	
POLS 1101	Am. Government	3	
HPS 1000	Fitness for Living	3	3 hrs

Lower Division Major Requirements			PREREQUISITE(S)
ACCT 2100	Introduction to Financial Accounting	3	ENGL 1101 & MATH 1101
ACCT 2200	Introduction to Managerial Accounting	3	ACCT 2100 & MATH 1106
CSIS 2300	Principles of Computing	3	NONE
CSIS 2301	Programming Principles I	3	CSIS 2300 & Credit Level Math
CSIS 2302	Programming Principles II	3	CSIS 2301
CSIS 2520	Introduction to Data Communications	3	CSIS 2301

Upper Division Major Requirements			PREREQUISITE(S)
CSIS 3210	Project Management	3	CSIS 2301
CSIS 3310	Database Design and Management	3	CSIS 2301
CSIS 3510	Organization and Architecture	3	CSIS 2302
CSIS 3530	Operating Systems	3	CSIS 2520 & CSIS 3510
CSIS 3600	Systems Analysis and Design	3	CSIS 3310
CSIS 4830	IS Integrated Project	3	CSIS 3600
CSIS 4840	Info Resource Management & Policy	2	CSIS 3600; take with CSIS 4841
CSIS 4841	IT Connections Lecture Series	1	CSIS 3600; take with CSIS 4840
ENGL 3140	Professional Writing in the Disciplines	3	ENGL 2110
MGT 3100	Management and Behavioral Sciences	3	60 credit hours
MATH 3400	Computer Applications in Statistics	3	CSIS 2300

Major Electives (Choose Six 3-hour classes)

Business Electives (Choose 1 – 3, no more than 3 from this area)		**PREREQUISITES**
ACCT 3100	Intermediate Financial Accounting & Auditing	ACCT 2100 & ACCT 2200
ACCT 3300	Accounting Information Systems	ACCT 3100
ACCT 4150	Auditing and Assurance	ACCT 3300
COM 3385	Organizational Presentation	COM 1109 or COM 1129
FIN 3100	Principles of Finance	60 Credit Hours
MGT 4160	Organizational Behavior	MGT 3100
MKGT 3100	Principles of Marketing	60 sem hrs of credit

Non-Business Courses (The remaining major electives are chosen from):		
CSIS 3150	Programming Languages	CSIS 3100
CSIS 3550	Unix Administration & Security	CSIS 2520 & CSIS 3530
CSIS 3401	Introduction to Data Structures	CSIS 2302
CSIS 4210	EDP Audit and Control	CSIS 3600
CSIS 4300	Web Development	CSIS 3600
CSIS 4310	Database Implementation Applications	CSIS 3310
CSIS 4400	Directed Study	Approval of Instr & Chair
CSIS 4420	Local Area Networks	CSIS 2520
CSIS 4490	Special Topics in Information Systems	Varies by Topic
CSIS 4500	Data Communication Protocols	CSIS 2520 and CSIS 3510
CSIS 4510	Computer Law	CSIS 3600
CSIS 4515	Computer Ethics	CSIS 3310 & ENGL 3140
CSIS 4555	e-Business Systems	CSIS 3210
CSIS 4575	Technology Commercialization	Any 3000 lvl Sci & Math
ISA 3100	Introduction to Information Security & Assurance	CSIS 2520
ISA 3200	Applications in Information Security & Assurance	CSIS 2520
ISA 3300	Policy & Administration Information Security & Assurance	CSIS 2520
ISA 3350	Computer Forensics	ISA 3100

Free Electives (Any courses in KSU curriculum totaling 12 hours)

Hours Required for Graduation

General Education	45
Lower Division Major Requirements	18
Upper Division Major Requirements	30
Major Electives	18
Free Electives	+ 12
TOTAL HOURS	**123**

- This form should only be used as a condensed summary. Please refer to the current KSU undergraduate catalog for complete details and official graduation requirements
- IS majors should enroll in major requirements as soon as possible. Do not complete your general education requirements before starting your CSIS coursework!
- Students wishing to graduate under this catalog must earn a grade of C or better in all major requirements
- Co-operative Education (CSIS 3396) and Internship (CSIS 3398) can only be used to fulfill free elective credit

03-04 IS REQ. Last modified July 7, 2003.

APPENDIX D

Computer Science Information Systems Undergraduate Major Sample Program of Study

2003-2004 Information Systems Sample Program of Study

Four-Year Program of Study for Full-Time Student

Status	1st Semester Program		2nd Semester Program	
Freshman (up to 30 hrs)	ENGL 1101	(3)	ENGL 1102	(3)
	MATH 1101	(3)	MATH 1106	(3)
	CSIS 2300	(3)	POLS 1101	(3)
	HPS 1000	(3)	COM 1109 of FL 2001 or PHIL 2200	(3)
	ECON 1100 or ECON 2100	(3)	CSIS 2301	(3)
	TOTAL HOURS:	**15**	**TOTAL HOURS:**	**15**
Sophomore (30 - 60 hrs)	ENGL 2110	(3)	HIST 1110	(3)
	CSIS 2302	(3)	ACCT 2200	(3)
	CSIS 2520	(3)	CSIS 3210	(3)
	SCI 1101	(4)	CSIS 3310	(3)
	ACCT 2100	(3)	SCI 1102	(3)
	TOTAL HOURS:	**16**	**TOTAL HOURS:**	**15**
Junior (60 - 90 hrs)	MATH 3400	(3)	MGT 3100	(3)
	CSIS 3510	(3)	CSIS 3530	(3)
	ART 1107 or MUSI 1107 or THTR 1107	(3)	CSIS 3600	(3)
	ENGL 3140	(3)	Free Elective*	(3)
	HIST 2112	(3)	Major Elective**	(3)
	ANTH 2105 or GEOG 2105 or PSYC 2105 or SOCI 2105	(2)	**TOTAL HOURS:**	**15**
	TOTAL HOURS:	**17**		
Senior (over 90 hrs)	CSIS 4840***	(2)	CSIS 4830	(3)
	CSIS 4841***	(1)	Free Elective*	(3)
	Major Elective**	(3)	Free Elective*	(3)
	Major Elective**	(3)	Major Elective**	(3)
	Major Elective**	(3)	Major Elective**	(3)
	TOTAL HOURS:	**15**	**TOTAL HOURS:**	**15**

* Prerequisites for electives vary by class. Check KSU catalog for current prerequisite requirements.
** Major electives are listed in the current KSU catalog. IS majors should consult their advisors to choose appropriate electives.
*** CSIS 4840 and CSIS 4841 must be taken together in the same semester – they are co-requisites.

This proposed schedule is not intended to reflect the availability of courses, but instead, is intended to demonstrate the feasibility of completing the B.S. in Information Systems by a full-time student in a standard four-year schedule. Each student should consult "Schedule of Credit Courses" for an accurate listing of course offerings. Part-time students should consider enrollment in the Summer Semester in order to stay on schedule for graduation. All students should plan their schedule around the major required courses, and begin their CSIS coursework as early as possible in their program of study.

Students should begin taking CSIS coursework as soon as possible and save free electives until the end of their program. Consult the KSU Undergraduate Catalog for degree requirements.

03-04 IS SAMPLE STUDY. Last Modified July 11, 2003.

APPENDIX E

Sample Syllabus of a Technical Writing Course at Kennesaw State University

TECHNICAL WRITING
Engl 3140-02: MW 8:00 p.m. - 9:15 p.m., HU 134
Engl 3140-03: TT 11:00 a.m. -12:15 p.m., HU 235

Instructor: Ms. Ruth A. Goldfine
Office: HU 107
Office Hours: T 1:00-3:00 p.m., W 3:00-7:00 p.m.

Phone: 770/423-6000 (Office)
770/xxx-xxxx (Home)
Email: rgoldfin@kennesaw.edu

Prerequisite: ENGL 2110

Texts and Materials:
- *Writing for the Technical Professions* by Kristin R. Woolever (Longman 2002)
- Companion Web Site: *Writing for the Technical Professions Online*: **http://wps.ablongman.com/long_woolever_wrtechprof_2**
- Online readings as assigned
- Web CT access (we will use WebCT for the online component of our course)
- Several computer disks (3.5-inch high density disks)

Recommended Materials
- A writing style guide (e.g., Diana Hacker's *A Writer's Reference*)
- A citation/documentation style manual *for your discipline*

Course Description

Welcome to Technical Writing! This course is designed to help you develop an effective method of planning and completing writing tasks so that you can meet professional writing demands. Most if not all technical professionals write on a daily basis in the workplace. Succeeding in the professional world requires not only technical knowledge but also effective writing skills; therefore, this course focuses on the writing skills necessary for advanced academic and professional writing, tailored specifically the technical fields.

In this course, you will complete several small assignments and lengthy final project. You will be participating in intensive writing, reading, revising, and/or peer commentary every week. Although the course will require much time and effort, by the end of the semester you will have produced several professional samples of your work that you can provide to prospective employers or graduate schools.

Upon successfully completing this course, you will understand how to:
- Analyze a rhetorical situation in order to develop documents that inform/persuade your readers
- Understand the cultural influences in the workplace that affect communication
- Identify and analyze audiences for particular types of writing
- Organize and present arguments effectively
- Examine sources in your field for their relevance and credibility
- Write effective memos, letters, short reports, and long, formal reports
- Edit your own work and that of your peers for content, organization, style, and mechanics
- Use graphics and page layout to support and enhance your written message and to create functional yet eye-catching communications
- Present your research findings to your peers

Course Requirements

- Attendance, participation, and deliverables noted below
- Weekly quizzes on the assigned readings (completed online)
- Deliverables:
  (1) A correspondence packet that includes one email message, one memo, and one letter
  (2) A 2- to 3-page source critique and annotated bibliography
  (3) A 3- to 5-page proposal and audience analysis
  (4) A PowerPoint presentation
  (5) Peer commentaries for other members of the class
  (6) An 8- to 10-page recommendation report on a topic related to your major or a collaborative website*
  (7) A resume and cover letter (optional; maximum of 3 pts extra credit)

*Students on a website production team must submit 2 additional assignments: (1) individual contract listing the tasks you will complete as part of the team, and (2) confidential evaluation after completion of the website, documenting how well you met the objectives set out in your individual contract.

Final Project

Content: The major project for this course is your choice of either (1) a 8- to 10-page analytical research report – not a typical "research paper," but a professional recommendation or analysis report – or (2) a website, created in collaboration with a *maximum* of 4 other classmates.

- Recommendation Report. You will use your research to support a personal position or recommendation related to your topic and within your field of study. You will select a real problem or situation to tackle, and your final report will be submitted to the appropriate individual (e.g., your department chair, your supervisor at work, etc.).

- Website. With a team, you will identify a client for whom you can develop a functioning website that will be implemented upon completion. You will not be paid for your services to the client, and you will be required to make a formal presentation of the website at the end of the semester.

Participation

Always come to class prepared, ready to contribute and take an active role in class discussions and activities. Your participation in peer reviews is especially important, both to you and your fellow students.

Grading

Written assignments will be graded on completeness of discussion and of assignment criteria, focus, organization, originality, effective argumentation and support, evidence of careful proofreading, correct grammar and effective sentence structure, among other criteria. I will calculate your final grade as follows:

Correspondence packet	10	%
Source critique and annotated bibliography	15	%
Proposal and audience analysis	15	%
Peer commentaries, participation, and other short assignments	15	%
Oral presentation	10	%
Final report or website with references	35	%
Total	**100**	**%**
EXTRA CREDIT: Resume and cover letter	3	pts

Course Policies

Attendance

Because this course includes an online component, we will meet in the classroom only once a week; the second scheduled weekly class meeting will be via the Internet. Consequently, attendance is required on those days we meet in the classroom. The only exceptions are documented medical excuses, religious holidays, and family/work "emergencies" (at the instructor's discretion). You are allowed 3 unexcused absences. For each subsequent unexcused absence, three points will be deducted from your final grade (i.e., if you have a total of 5 unexcused absences, 6 points (3 points for each unexcused absence beyond the 3 allowed) will be deducted from your final grade. If you miss a class, *you are responsible for obtaining the information, course changes, and/or materials that were covered or distributed the day of your absence.*

Assignment Deadlines

You must turn in or have all assignments ready at the start of class on the day they are due. **NO late assignments are accepted** (i.e., you will receive a "zero" for the assignment). If you must miss a class on a day an assignment is due, turn in the assignment to me early, arrange to have a classmate turn in your work at the beginning of the class period it is due, or email the assignment to me PRIOR TO the scheduled start time for our class. Do not assume that I have received your emailed assignment until I have replied to your email message. It is your responsibility to follow-up with a call to my office or home if you do not receive a reply from me prior to the start of that day's class. "The computer ate my file" or "My printer wouldn't print my paper" are not legitimate excuses.

Format of Assignments

All work must be typed or word-processed and printed with a **readable** printer. **I will only accept assignments via email on a case-by-case basis. You must have prior approval from me before submitting an assignment by email.**

Papers should have one-inch margins on all sides (top, bottom, left, right) and font size must be 10-, 11-, or 12-point type in a professional-looking font style (no Courier or Courier New). All papers must be double-spaced unless otherwise noted on the assignment. Visual aids (i.e., graphics) do not count toward the page-length requirement for the assignment.

Keep copies of all work on floppy disks, hard disks, and paper. Do not throw away any of your returned work, especially drafts of the final report with my comments. Staple pages together if an assignment is more than one page, and include your name and page number on all pages of each assignment.

Class Cancellation

If classes are canceled for any reason, you are responsible for completing the assignments scheduled for that day and for preparing properly for the next class as outlined in the daily schedule. If there are any changes to be made to the schedule, I will send an email message to all students.

Plagiarism

When you use the citable work of someone else, document your source. If you use someone else's words or ideas without acknowledging the source, or if you attempt to deliberately represent someone else's ideas and/or written work as your own, then you have committed plagiarism, a serious breach of academic and professional conduct. If you plagiarize, I am bound by university policy to report your actions to a review board. Penalties range from failure of the individual assignment to failure of the course and, in some cases, suspension or expulsion from the university. Be sure to use correct documentation format and include a references page with your papers (we will talk more about documenting sources in this course). If you have questions or concerns about plagiarism as you complete your assignments, please ask me.

COURSE SCHEDULE
ENGL 3140-03: TECHNICAL WRITING
Mon/Wed HU 134

			Readings & Projects Due
Aug	18	Course Introduction / Review Syllabus; Choosing a topic for the final project; Review WebCT and companion website; In-class writing sample sent as email attachment	
	25	**Online:** Read Chapter 1 and take the online quiz at our companion website. Email the results to me at: *rgoldfin@kennesaw.edu*	Ch. 1
	27	Discussion of final project topics; Discuss writing email messages; *Assignment: Final Project*; Helpful resources: http://web.ics.purdue.edu/~blankert/420/corporate/recommendationreport.html http://www.io.com/~hcexres/tcm1603/acchtml/recomx2a_non.html http://www.io.com/~hcexres/tcm1603/acchtml/recomx4a_non.html http://ocean.otr.usm.edu/~wsimkins/recommendation.html http://www.beresourceful.org/downloads/casestudy_semico.pdf http://www.lhmu.org.au/lhmu/campaigns/smoke_free_03/files/Smoking%20in%20Hospitality%20RECOMMENDATION%20REPORT.pdf Send me an email identifying your final project topic and team members (if applicable)	Ch. 6; revise & send sample
Sep	***01***	***NO CLASSES – LABOR DAY***	
	03	Discuss organizing for readers; *Assignment: Correspondence Packet*	Ch. 3
	08	**Online:** (1) Complete online quiz for Ch. 3 and email the results; (2) Read both online articles for Ch. 3 (at our companion website) and be prepared to discuss them in class on Tuesday	
	10	Discuss Ch. 3 online articles; Discuss and perform peer reviews of correspondence packets	Corr. Pk. (Draft)
	15	**Online:** (1) Read Ch. 2 and complete online quiz; email results; (2) Read the online article "The Quality of the Writing Can Never Be Better than the Quality of the Research" at www.raycomm.com/techwhirl/phpapps/pfv/pfv.php?/techwhirl/magazine/writing/hindsights_qualitywriting.html	Ch. 2
	17	**Discuss online article for Ch. 2**; Researching and critiquing sources; Assignment: Source Critique and Annotated Bibliography	Corr. Pk. (F) Bring journal, from your disc.
	22	**Online:** (1) Submit via WebCT a *draft* source critique (1-2 pages) for one of the journals you will use for your final project; use the "Source Critique Worksheet" posted on WebCT to assist you in Writing your critique; (2) Visit and READ the following websites: http://www.library.cornell.edu/okuref/research/skill28.htm http://www.crk.umn.edu/library/links/annotate.htm	

Sep	24	Discuss topics, articles, and annotated bibliographies Locate and share online resources for formatting sources	Bring 3-5 articles for your project
	29	**Online:** (1) Read Ch. 5 and complete the online quiz; email results; (2) Read and write a 1-2 sentence summary of the four online articles for Ch. 5; submit summaries via WebCT	Ch. 5
Oct	01	Discuss editing and online articles Conduct peer reviews of Annotated Bibliographies and Source Critiques	Annotated Bib & Source Crit
	06	**Online:** Read Ch. 14 and the following online articles: www.raycomm.com/techwhirl/magazine/writing/inspectionmethod.html www.fastcompany.com/online/02/meetings.html Write a brief one-page reaction, stating which elements you found useful and would hope to incorporate into your class and/or professional work. Submit via WebCT	Ch. 14
	08	Creativity Day	**Annotated Bib & Source Crit**
	13	**Online:** (1) Read Ch. 11. (2) Examine the following websites. Select ONE, and write a one-page critique. Bring the critique to the next class meeting. http://fbox.vt.edu/eng/mech/writing/workbooks/proposals.html http://writing.colostate.edu/references/documents/proposal/index.cfm http://www.emory.edu/EDUCATION/mfp/proposal.html ***Last day to withdraw without academic penalty***	Ch. 11
	15	Review audience analysis (Ch. 1, pp. 13-17) Discuss proposals / Share website critiques In-Class Writing: Write a *draft* audience analysis and outline proposal *Assignment: Proposal and Audience Analysis*	**Crit of Website** Aud. Analysis (Draft)
	20	**Online:** (1) Read Ch. 8. (2) Review the first 3 online Web Resources for Ch. 8; print information relevant to your final project (Note: the Jerz links are not functional)	Ch. 8
	22	Discuss descriptions and summarizations Peer review	Proposal & Aud. Analysis (D)
	27	**Online:** Take online grammar quiz	
	29	Discuss reports and studies Present proposals to class	**Proposal & Aud. Analysis (F)**
Nov	03	**Online:** (1) Read Ch. 9 and complete the online quiz. (2) look over the following web sites: *abstracts:* http://owl.english.purdue.edu/handouts/pw/p_abstract.html http://www.io.com/~hcexres/tcm1603/acchtml/abstrax.html http://www.rpi.edu/dept/llc/writecenter/web/abstracts.html http://www.biogeog.ucsb.edu/projects/ibm/ibm_pubs.html *executive summaries:* http://writing.colostate.edu/references/documents/execsum/index.cfm http://www.hud.gov/library/bookshelf18/archivedsum.cfm	Ch. 9
	05	In-class work on final projects / Discuss abstracts and executive summaries	

Nov	10	**Online:** Read Ch. 13 and look over the following sites: http://www.ruf.rice.edu/~riceowl/oralpres.html http://www.ukans.edu/cwis/units/coms2/vpa/vpa.htm http://www.kumc.edu/SAH/OTEd/jradel/effective.html http://fbox.vt.edu/eng/mech/writing/workbooks/visuals.html	Ch. 13
	12	Discuss presentation Conduct "practice" presentations	
	17	**Online:**(1) Read Ch. 7 and the Ch. 7 online article, "Online Technical Writing: Instructions, at: http://www.io.com/~hcexres/tcm1-603/acchtml/instrux.html (2) Bring short set of written instructions to next class meeting	Ch. 7
	19	Critique instructions A demonstration *Assignment: Instructions (a presentation)*	**Final Projects**
	24	**Online:** Read Ch. 17 and submit online quiz OPTIONAL: Email DRAFT resume and cover letter to me for review	Ch. 17
	26	***NO CLASS – THANKSGIVING BREAK***	
Dec	01	**Online:** Submit confidential evaluation via WebCT OPTIONAL: Email FINAL revised resume and cover letter to me for extra credit	
	03	***PowerPoint Presentations***	**Presentations**
	10	**FINAL EXAM – 8:00 p.m. - 10:00 p.m.** ***PowerPoint Presentations***	**Presentations**

Calendar is subject to change.

APPENDIX F

Demographics Questionnaire

Personal Information

1. Name or identifying number: ______________________________________
*Last four digits of social security number followed by eight-digit birthday: 123401181982

2. Sex: Male Female

3. Date: ________________________

4. Ethnicity/Race:

African-American	Caucasian	Asian
Hispanic or Latino	American Indian or Alaska Native	Native Hawaiian or other Pacific Islander

Other: ________________________

5. Which age category best describes your current age:

18-22 23-28 29-35 36-4546+

6. Is English your second language? Yes No

If YES, what is your first language (i.e., your native language)? ____________________
____

Educational Information

7. What is your Major: CS CSIS Other ______________________

8. Year: First-Year Sophomore Junior Senior

9. Other Writing Courses Completed at Kennesaw State University:

English 1101 Composition I
English 1102 Composition II
Other ______________________________________

10. Other Writing Courses Completed at a university, college, or community college other than Kennesaw State University: (please list)

11. How do you rate your own writing ability?

Excellent Good Adequate Fair Poor

Employment Information

12. Are you currently employed? Yes No

13. If employed, are you employed in your field (e.g., CS, CSIS, etc.) Yes No

14. If employed, are you required to write as part of your job? Yes No

15. Income Level: < $19,000/yr $19,000–30,000/yr $30,001 – 45,000/yr >$45,000/yr

16. For how many years have you been writing as part of your job?

0 – 2 years 3 – 5 years 6 – 10 years 10+ years

17. If you are required to write as part of your job, which of the following do you write:

letters	proposals	reports (final, interim, travel)
emails	website text	users manuals
memos	instructions	Other (please list):

18. If you are required to write as part of your job, which of the following do YOU believe you are able to write well: (circle ALL that apply)

letters	proposals	reports (final, interim, travel)
emails	website text	users manuals
memos	instructions	Other (please list):

19. If you are required to write as part of your job, which of the following items do YOU believe you write poorly: (circle ALL that apply)

letters	proposals	reports (final, interim, travel)
emails	website text	users manuals
memos	instructions	Other (please list):

Computer/Computer Programming Knowledge

20. In which computer programming languages are you fluent:

Java	HTML	PROLOG	FORTRAN	C++	Ada	Pascal
LISP	Perl	ML	COBOL	ALGOL	Visual BASIC	

21. With which programming languages are you familiar and/or minimally capable:

Java	HTML	PROLOG	FORTRAN	C++	Ada	Pascal
LISP	Perl	ML	COBOL	ALGOL	Visual BASIC	

22. How do you rate your computer programming skills?

Excellent	Good	Adequate	Fair	Poor

23. How do you rate your word processing skills?

Excellent	Good	Adequate	Fair	Poor

24. Please list the software packages with which you have expertise:

Word Processing:

Spreadsheet:

Database:

Desktop Publishing:

Web Development:

Graphics:

Others:

25. Please list the software packages with which you have basic competence/familiarity:

Word Processing:

Spreadsheet:

Database:

Desktop Publishing:

Web Development:

Graphics:

Others:

26. Would you be willing to participate in a brief interview (10 – 30 minutes) at the end of this semester regarding this course?

YES NO

APPENDIX G

Pre-Test Self-Reporting Questionnaire

Name/Identifying Number: ____

Date: __________________________

For each statement below, please circle the number that best describes your degree of agreement or disagreement with that statement, then write in a response to the directions that follow each statement.

	Disagree Strongly				**Don't Know**					**Agree Strongly**
1. My writing skills may improve during this course. *Please list the specific areas in which you believe your writing skills may improve*:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2. My existing knowledge base may contribute to my ability to master the writing skills and strategies presented in this course. *Please list the specific knowledge you believe will assist you, if any, in mastering the writing skills and strategies that will be presented in this course*:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
3. My understanding of computer languages may assist me in mastering the strategies of effective technical and professional writing. *Please list those computer languages, if any, that you believe your understanding of will assist you in mastering the writing skills and strategies that will be presented in this course*:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
4. I believe many strategies for writing may be similar to the strategies used to write computer code. *Please list the writing strategies and the coding strategies that are similar*:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

5. I believe I might better understand the rules of grammar and the theories of composition if the instructor used analogies related to programming languages to make connections that relate to me. *Please list any analogies you believe would assist you in better understanding the rules of grammar and the theories of composition*:	1 2 3 4 5 6 7 8 9 10
6. I believe the rules of grammar may be similar to the rules that govern computer programming. *Please list the rules of grammar that you believe may be similar to the rules that govern computer programming:*	1 2 3 4 5 6 7 8 9 10
7. I believe there is little or no similarity between computer programming languages and the English language. *Please list the reasons you agree or disagree with this statement.*	1 2 3 4 5 6 7 8 9 10
8. The writing process I use to write an essay may change because of the writing strategies and theories I will learn in this course. *Please list the ways, if any, in which you think your writing process may change.*	1 2 3 4 5 6 7 8 9 10

APPENDIX H
Post-Test Self-Reporting Questionnaire

Name/Identifying Number: ______________________________

Date: ____________________

PART I: *For each statement below, please circle the number that best describes your degree of agreement or disagreement with that statement.*

	Disagree Strongly				Don't Know					Agree Strongly
1. My writing skills improved during this course.	1	2	3	4	5	6	7	8	9	10
2. My existing knowledge base contributed to my ability to master the writing skills and strategies presented in this course.	1	2	3	4	5	6	7	8	9	10
3. My understanding of computer languages assisted me in mastering the strategies of effective technical and professional writing.	1	2	3	4	5	6	7	8	9	10
4. I believe many strategies for writing are similar to the strategies used to write computer code.	1	2	3	4	5	6	7	8	9	10
5. I believe I would better understand the rules of grammar and the theories of composition if the instructor used analogies related to programming languages to make connections that relate to me.	1	2	3	4	5	6	7	8	9	10
6. I believe the rules of grammar are similar to the rules that govern computer programming.	1	2	3	4	5	6	7	8	9	10
7. I believe there is little or no similarity between computer programming languages and the English language.	1	2	3	4	5	6	7	8	9	10
8. The writing process I use to write an essay changed because of the writing strategies and theories I learned in this course.	1	2	3	4	5	6	7	8	9	10

PART II: *Please respond to each question or statement below in the space provided.*

1. If knowledge of a specific computer programming language assisted you in mastering the writing skills and strategies presented in this course, please list those languages.

2. Which specific elements of the computer programming languages listed in item 1 in particular were useful to you?

3. Having nearly completed this technical writing course, what similarities, if any, do you recognize between computer programming languages and the English language?

4. Having nearly completed this technical writing course, what similarities, if any, do you recognize between the rules of grammar and the rules that govern computer programming?

5. Having nearly complete this technical writing course, what similarities, if any, do you recognize between the strategies used in composition and the strategies used in computer coding?

6. Having nearly completed this technical writing course, how could the instructor have used analogies relating to computer programming languages and coding to enhance your understanding of the rules of grammar and theories of composition?

7. How has your approach to composition been affected by your knowledge and understanding of computer languages and computer programming, if at all?

8. How has your approach to computer programming been affected, if at all, by the writing strategies, grammatical rules, and composing process presented in this technical writing class?