Teaching Writing in Computer Science

A. Tircuit

1 Computer Science, Seton Hill University, Greensburg, Pennsylvania, USA

Abstract – Writing is an important skill for all graduates to master, yet it is often difficult for CS professors to embrace writing due to the inherent difficulties in teaching writing and CS faculty’s lack of experience with writing based pedagogies. Teaching writing can be done well as a complement to a curriculum, providing both added value to learning and valuable skills for the future. In this case, writing was added to a Programming Languages course. As the students designed their own programming languages, they were required to provide a language specification and analysis of features. Using creative pedagogies, writing instruction was added to the course without neglecting content. Many writing techniques were also adapted to be suitable in a Computer Science course.

Keywords: writing intensive, undergraduate CS education, programming languages

1 Introduction

I would rather teach students to write in C++, Java, or even assembly language than English. Really. Yet writing is recognized as a vital skill by both many computer science factions, and at my university. The ACM includes writing as a part of its CS curriculum guidelines [1]. More recently writing has been a subject of CS education research [2, 3, 4, 5, 6]. And as Dugan relates in his experiences involving writing in the technology workplace [4] a skillful writing ability is an asset to any technology professional. The writing intensive (WI) wave that swept through Seton Hill University (SHU) in spring of 2006 was the final motivation I needed to get past my squeamishness of teaching writing. Our Computer Science program needed a writing intensive course. I had the motivation and resources to turn my existing writing assignments into something meaningful both to fulfill this requirement and enhance the students’ education. For better or for worse, I was caught up in the writing current. As you can probably imagine, the ride was both more and less difficult than I feared. The journey has been worthwhile however, as it has both improved my course and added value to the students’ education.

1.1 Contents

The remainder of this paper is organized as follows:

2. Attempting to please two masters: CS meets writing intensive
3. Writing as a process
4. Results
5. Future plans
6. Current research in teaching writing in computer science
7. Conclusion
8. Acknowledgements
9. References

2 Attempting to please two masters: CS meets writing intensive

Given that my Programming Languages course already featured a large writing project, it seemed like the logical place to implement the WI initiative. It also fit several of the requirements of the WI committee: it is a required class for all CS majors, is taught to sophomores or higher leveled students, at which point the students are expected to be ready for advanced writing instruction. With high hopes, I submitted my WI proposal.

Unfortunately, the WI committee did not speak “computer geek,” and I fail at “WI language” more often than not. My first proposal was rejected, on the grounds that it did not meet many of the requirements for a WI course. Describing my writing project as the process in which students choose a software domain and design a programming language to best-fit that domain was less than helpful to the WI committee. I did not do such a great job of interpreting their requirements in the spirit they had written them, either. Fortunately we geeks stick together, and I enlisted the help of a friend whose specialization is in technology combined with journalism. He certainly knew the writing part, and he knew whose specia-

lization is in technology combined with journalism. He certainly knew the writing part, and he knew enough of my language to help me translate my proposal into something the WI committee would understand. I continue to be grateful for his help. Dugan utilized a similar tactic when designing his writing course. However he used a slightly different pedagogical model than ours at SHU by team teaching the course with a writing program specialist [4]. Our model insists that the experts in each discipline teach writing specific to that discipline.

Once I sufficiently understood the WI requirements, the second challenge was to incorporate them in to my course, without sacrificing anything I thought was critical to the curriculum from a CS prospective. This continues to be the challenge today. Programming Languages has tons of content,
enough to easily fill an entire semester. At the same time, I now had to incorporate pre-writing exercises, peer review, and 20 pages of submitted and revised writing by the students. If there were one thing I did really poorly the first semester, this would be it. Three attempts, and several creative ideas later, the goals of teaching content and writing are much closer together.

2.1 The writing assignment

At the heart of my WI class is the “design your own programming language” writing assignment. Students are instructed to take the principles they learn in Programming Languages, and design a language of their own. Students pick the domain, make a plan for development, and choose features for their language. Students are required to justify their choices, either by domain suitably, programming language characteristics, technological necessity, or some combination. “Because it looks pretty” is not an acceptable reason for choosing a language feature. “Because it is the most suitable option for my domain” is.

Pre-WI, one of the mistakes I made was in not sufficiently specifying the learning objectives for the writing assignment. It was a way to evaluate the students’ analytical ability, but not much else. Without these specific guidelines, variance existed in abundance in the quality of students’ assignments, and it left me without an appropriate mechanism for evaluation. Teaching is a learning process for me as well, and this is one thing I learned to do better. Now the assignment has a defined audience: a person or organization who will fund the development of your programming language, if it should be sufficiently well designed and you have expressed your ideas clearly and with strong rationale for your language choices. Furthermore, I have a goal for the students: learn to write well enough to write a proposal (email or formal) at your future place of employment. Given these clear objectives, the students’ work has become better defined, and I have an improved assessment rubric.

One question that often occurs is concerning my decision to have the students practice writing a proposal instead of a research paper. Seton Hill University’s CS program is an undergraduate-only program. A few of our graduates aspire to go to graduate school, however most see themselves working in the CS industry. Teaching writing as they will most likely use it in their careers is both a logical choice, and an “easier sell” for the students. I am able to critique them with the question, “Would you write the argument this way if you were writing this for your boss?” providing both a framework for the writing assignment and meaning to the effort of completing the assignment. The tradeoff is that research methods are not a focus. Sources are required in sufficient quantity to make your argument.” Were we a graduate level program, or if we attracted a different population of students, this lack would have to be addressed. The “design your own programming language” assignment also has the benefit of being closely tied to the content of the course. A portion of the course content is explored through writing, as suggested in [2]. This frees up time during class for writing instruction and writing exercises such as peer evaluation. This strategy has made increasing writing in a content-heavy course quite manageable.

3 Writing as a process

I firmly believe that writing is a process. I teach programming using that methodology [7, 8], and I also use a process model in my WI course. Of course, I am not alone in using a process-based pedagogy [9]. And as [6] mentions, getting students to engage in the process is often difficult. My strategy is to require the students to submit a variety of prewriting exercises and drafts, enforcing at least some forethought and revision.

3.1 Prewriting

Prewriting is an essential step in the writing process. When addressing prewriting for my WI course, I returned to my roots in developing pedagogy for programming classes. [7, 8] Prewriting activities are brief, worth only a small percentage of the grade, and feedback is provided immediately via class discussion. Most often, I grade these on a participation basis. Class discussion has an additional benefit: it gives the students input from their peers as well as from me. The one exception is the final prewriting assignment in which students define the domain and main goal(s) for their language. This acts at the thesis statement. I carefully evaluate these to ensure the student has a focus before continuing his/her paper.

The domain of the language becomes the theme around which all other decisions are built. A student who wants to write a language for mobile computing must address issues such as working in a networked environment and portability of code. A student choosing the domain of video game development will likely spend time exploring helpful language elements when creating game engines and processing graphics. Another popular domain is beginning programmers. These students will explore issues with readability and writability on a higher level. Students who choose “general purpose” as their domain have the greatest difficulty in making language decisions later on. Many general-purpose languages exist, and are used quite commonly [10, 11]. It is difficult to come up with new features for something that is already well studied.

3.2 Drafts

For each draft the student is required to make a variety of decisions about his or her programming language. The first part of the assignment focuses on overall decisions: domain, criteria, important characteristics, and plans for development. As we cover programming language features in class, the students must decide which ones they want to add to their languages. Students must decide upon variable types, control structures, data structures, and advanced concepts such as object orientation. For each decision, the student must define
the rationale behind it, often via analyzing the features of existing programming languages and their resulting impact. It is not sufficient to say, “I chose to include object orientation because C++ does it and I like C++.” An appropriate argument would discuss the need for organizing large projects, writing reusable code, and benefits of using an object model as a design tool (modeling programming objects after real life).

I expect the first draft to have complete thoughts and paragraphs focused around each subject. Students should have already searched for any sources they need and provide proper citation of sources. In some cases this may require a student to do some revision before they submit each draft (the drafts are submitted in parts, and the four parts combine to make the final paper). It is up to each student ascertain this for his or herself.

I evaluate drafts first on the strength of the argument each student makes for his/her language choices. I will point out places where the student does this particularly well or particularly poorly. I will note anything that I find unclear or contradictory. In keeping with the practices of English composition experts, I look at grammar and sentence structure as secondary issues, except in the case where they inhibit understanding of the paper.

It is often said in hockey that a penalty is only a penalty if the referee notices. My rule of thumb when evaluating grammatical issues is the same: “it’s only an error if I notice it’s wrong.” My grammar and proofreading skills are sufficient to catch gross mistakes. They will not catch every detail. I find this to be an acceptable compromise. I do not want to be a “grammar police” as it would distract from the primary focus of building a strong, consistent argument. At the same time, students will be expected to be able to write reasonably well in their future employment. It is in this vein that I note significant grammatical issues and require the students to fix them.

I find that it is not helpful to inform the students of my rule of thumb. Unfortunately, many seem to believe that if they do not notice a grammatical error, I will not either. This is not the case, particularly when everyday speech (at least in this area) has a noticeable lack of basic grammatical principles, such as subject-verb agreement. Instead I instruct the students to use grammar and language that they would find appropriate in a work environment. When a draft has particularly poor grammar, I refer the student to our writing center for expert assistance.

I hate page lengths. Were I given complete free rein, I would have a list of requirements, without any mention of how many words it should take a student to fulfill them. The WI committee however feels it is necessary to require 20 pages of finished work. (The 20 pages may be divided into multiple assignments.) I give the students some leeway in their final page count: 18 pages is sufficient for full credit. That is about as much as I can stretch that rule though and still stay within the WI guidelines.

In order to meet the page count, many students repeat themselves, add unnecessary quotes, and use creative formatting techniques. My blanket policy is: if your paper is not long enough, find something else meaningful to add to it. I take off points for any extraneous text or creative formatting. While I do not agree with a mandatory page-length, if we are going to have one, we are going to address it properly.

3.3 Final Submission

The final submission includes an executive summary, the paper itself, and a list of sources. As noted in [4], CS does not have a single style format for all publications. As spending time on formatting and style is not a priority for me, I give basic guidelines: 1-inch margins, 12 pt font, any recognized citations style. Most students choose MLA, with which they are familiar [12]. Should a student express interest in using a CS-specific style, I would suggest the ACM publication guidelines. When grading the final version I specifically look for corrections and improvements for any issues I noted in the drafts. This process is streamlined by our online course management system. Students upload digital versions of all assignments. When grading each draft, I make comments in the system. These comments are both immediately available to students and preserved for future referencing by me when grading their final papers.

4 Results

Implementing the writing assignment was relatively straightforward. The challenge comes in evaluating student product to make the WI elements stronger in the future.

4.1 Students write both better, and worse, than I give them credit for

One of the challenges of a small university is that many of our upper division courses, including Programming Languages, is only taught once every 2 years. Traditional 4-year students will take Programming Languages in their junior or senior year. They will have completed 2 semesters of programming and either have had data structures previously, or be taking it concurrently. Quite a few of our students instead follow what I call “the three year plan.” That is, they are attempting to meet 4 years of CS requirements in 3 years. It can be done, but it often means a student takes upper-division courses without the ideal prerequisites. If I were a dragon insisted that all prerequisites be met before entering my courses, we would lose majors. Since retention is also critical, I do the best I can to work with variety of preparedness of my students. The only guarantee is that every student has successfully completed CS1, meaning they have basic programming skills.

After several years of running the WI class it has become apparent to me that the writing assignment is not challenging to the better-prepared students. It is relatively simple for someone with sufficient background knowledge to analyze the language information we discuss in class, to seek
out additional information where necessary, and to form their language goals into coherent sentences. Some of these students surprise me by going above and beyond the assignment instructions. They will address higher-leveled concepts without my specific direction to do so. I am always impressed by a student who goes out of his or her way to gain worthwhile knowledge from an assignment, even if I have not done a particularly good job of requiring them to do so. Some of the “better than average” students write papers that adequately meet the goals of the assignment, and express their boredom with the topic.

The obvious answer of making the assignment overall more challenging is not necessarily the best one. While some students surprise me with the strength of their papers, I am also surprised by the inadequacy of some submissions. For example, many students stated on their initial drafts that they wanted their language to be simple and offer many advanced features. Wouldn’t we all! A second irresolvable conflict is the tradeoff between efficiency and reliability. A programming language with built-in structures to ensure reliability is unlikely to be as efficient as one without them (efficiency of executable code). Yet many students make these two goals their primary concern. We discuss these issues in class, they are well defined in our textbook [14], yet they are persistent and make it into many papers.

4.2 Peer review… sort of

I have bad experiences with peer review. When the WI committee listed this as one of their criteria for a WI course, I cringed and vowed to do it as little as possible.

Writing pedagogical experts consider peer review to be essential as it teaches students to work collaboratively, learn how to give advice, and learn how to take advice. In many cases, the purpose of peer review is not the outcome, but the process. Students who are struggling might review papers that are better than their own, giving them examples of what they could accomplish. Students who are already effective writers get the chance to practice evaluating ideas of less adept writers [13].

I have been slow to adopt true peer review. Instead I have focused on peer collaboration. After each draft submission I select a number of quotes from student papers that illustrate the most frequent and most serious issues. I copy and paste the quotes into a blank document, and remove anything that could identify a particular student. In class, I will put these quotes on the projector, and we discuss them as a group. This is not a graded activity. I frame it as an opportunity to assist one another in improving their writing skills.

One example from a student paper, when discussing reserved words and keywords:

“To start out, I feel like a well-structured language needs to have a collection of words that are set aside to perform a certain action. It increases reliability also makes it more readable. In my opinion, there’s really no reason to have keywords where a variable could potentially have the same name. That leaves so much room for ambiguity and ambiguity does not sit well with me. Although it might cut down on the flexibility of the language, I think it is a minor and acceptable compromise to ensure that reserved words do not get confused with variable names and such.”

I often ask questions to stimulate the discussion. Questions I have asked about a paragraph such as this one:

1. What language issue is the author talking about?
2. What argument is the author attempting to make?
3. Is that argument clear from the text? How could we define it better?
4. Is the language formal enough for our purposes?

In this case, the students reported that they were unsure what the author was attempting to discuss. Upon hearing that it was concerning keywords verses reserved words in a language, they pointed out that the author did not state his argument well – the term reserved words was not used. They also expressed concern with the over-use of the word “I”, and of personal opinion. Phrases such as “does not sit well with me” were noted for their lack of formality.

Another example of a quote from a student’s paper:

“Primitive data types are defined as “not being defined in terms of other data types” (Sebesta, 250). XXXX will have only three primitive data types that can be used while programming. They are going to be the types of Integer, Boolean, and Char. These types are going to help XXXX achieve the goal of being simple to use. For the types of programs that are going to be written with XXXX there is no need to have many primitive data types in the language. Integer, Boolean, and Char will allow the new programmer to write effective code without causing confusion on which type to use. With XXXX there is only three data types Integer for storing values in whole number and has several sizes; and Boolean for returning true or false, and Char for strings; this will keep simplicity (Sebesta). A disadvantage of only having these three types is that values will not be able to have decimal places with using a float data type. The Advantage of only using these three types and not decimal is that decimal does not use memory effectively and it has a restricted range (Sebesta).”

In this case students pointed out that the author’s language had a higher level of formality. The author cited the text, and made a reasonable attempt at making language decisions based on his goal of making the language simple for new programmers. The students however also felt that the author’s decision to eliminate floating-point variables was going to limit the language and was not well justified. In particular, I raised the question, “Do you think kids will be learning programming before they are learning math with decimal places? If there are no doubles, can a kid do something like compute sales tax on a pizza? Do you think that’s a reasonable expectation for someone learning programming?”

It was decided that omitting floating-point variables (and
languages are used to analyze the impact of feature choices. For example, a language that is intended to be general purpose may be compared to C++. A language intended for teaching purposes may be compared to Pascal, BASIC, or even the Scratch programming environment [15]. (Scratch has many similar features to a good beginner’s language, despite its graphical format.) This process ensures the students’ language decisions are well made, and requires using analytical skills that will be useful in the future.

Perhaps the most obvious benefit of teaching writing is that it provides an additional way to assess student learning. I discovered the benefit of this when repeated papers that incorrectly interpreted a passage from the text. The following passage refers to the AND and OR operators available in C++, and their short-circuit characteristics [14]:

In the C-based languages, the usual AND and OR operators, && and ||, respectively, are short-circuit. However, these languages also have a bitwise AND and OR operators, & and |, respectively, that can be used on Boolean-valued operands and are not short-circuit. Of course the bitwise operators are only equivalent to the usual Boolean operators if all operands are restricted to being either 0 (for false) or 1 (for true).

The students used it to claim that the && and || operators are overloaded to have multiple functions. A traditional programming languages assignment or test would never catch that error, which became obvious when evaluating writing assignments. During the following class period I followed up my comments on student papers with a discussion of that passage and the various ways it had been interpreted by the students. This not only assisted the students in learning the specific differences between the & and && operators, it also provided an immediate motivation for the students to learn how to read and understand academic materials.

5 Future Plans

In particular, addressing the boredom level of the more advanced students is of particular concern. One approach I am considering is to come up with a second level to the assignment. The first level would allow anyone to complete it and to obtain a passing grade. The second level would require more depth both in obtaining sources and building arguments. Students may be required to study the available programming languages beyond the ones we see most commonly. Students might be required read a reference written on a higher level and display their understanding by using it in their paper. Or students could be asked to provide additional information concerning their language such as BNF notation for some of the structures [16].

Another persistent issue is that many students basically recreate an existing programming language. Some students are attempting to stay in their comfort zones. Some students enjoy using a programming language they are familiar with, and do not see any need for a different one. One of my goals

subsequently floating point math) is not necessarily a terrible decision, however it was not sufficiently justified by the author. Two issues remained that the students did not uncover. One was that the author mixed up discussion of floating point type with that of decimal type. The comment concerning inefficient storage and limited range of decimals is applicable given a decimal type stored as a string. It is not relevant to the most common floating-point type. The second involved the academic level of the paragraph. The target audience for the paper has experience with programming and programming language theory. It is not necessary that the student explain basic concepts such as the purpose of an integer. I suggested that the student would be more likely to lose the interest of his audience if he kept such basic information rather than focusing on the analysis of language decisions.

One of my goals of the peer collaboration exercise is to teach students to critically evaluate the feedback they are given. Initially, I do this. If a student says that a passage “doesn’t make sense,” I may respond with “How does it not make sense?” If a student says they like another students idea, I will ask, “What do they like?” By the end of the exercise, both the initial feedback and responses to it should come from the students.

4.3 Enhancing student learning

Writing is a valuable skill for all graduates. However, simply teaching writing would be a poor use of our time. A writing assignment may often be used to require the students to develop a deeper understanding of the course content by requiring higher-order academic skills such as synthesis and analysis. In doing this, we provide a higher value to student education, and implement courses on a level that is consistent with the expected abilities and the level of product required of upper division students.

Programming Languages is a content-heavy course; perhaps the most content-heavy of any I teach. It would be simple to quiz students on their ability to memorize the material. What little problem solving is inherent to the course is limited to the units on lexical analysis and parsing. This is far less critical thinking than I require in my upper division courses. Adding the writing component to this course in particular provides educational value in that it greatly increases the higher order thinking skills required of the students. Throughout the process of designing their own programming language, students must synthesize various concepts taught in the class. At each stage of the paper students must refine their language and increase its features. As students add additional features, they must analyze their choices on multiple levels. The first level is that of their language: the language features should unify to produce a consistent, cohesive programming language. The second level of analysis requires students to look at their language choices in relation to programming language principles at large. They must ensure they have designed a language that is capable of being implemented. Examples of existing programming
with the Programming Languages course is to get students out of their programming comfort zones! Designing your own language is a great way to do that. Given the product of the students so far, it appears I am going to have to find a way to make that a requirement, not just a suggestion. I am still looking for creative ways to accomplish that beyond just saying, “Choosing a language feature because it is like your favorite language is not an acceptable argument.”

My understanding of peer review has grown as I have worked on the course. I went into teaching writing with the belief that peer review was a way faculty passed some of their work onto their students. I have a better understanding now that it is about the students learning and engaging in the writing process. I am currently working on the details of a true peer review assignment that I am going to add to the course.

My intention is to continue to improve both my own skills in teaching writing and the writing instruction provided to my students. While the “design your own language” writing assignment could in theory be used indefinitely, I also plan on exploring additional options for the course. One possibility is to have students research a cutting-edge programming language, and perform an analysis on the new language.

6 Current research in teaching writing in computer science

Anewalt speaks of teaching writing in CS for the first time in her paper [2]. Her experiences mirror my own in that she found herself teaching at a university that requires writing across the curriculum. She addresses a number of fears faculty face when embarking on the journey of teaching writing. She addresses the ever-present concern that adding to a course takes time away from content, the suitability of writing in various courses, the time necessary to grade writing-based assignments, and the situation that faculty of various disciplines face when they attempt to teach a subject out of their comfort zone. I found her suggestions on the design of assignments to be particularly helpful. I find her insights to be on-target with my own experiences, and her suggestions applicable to a wide variety of writing situations. The feedback she gathered from students’ notes that writing assisted them in developing an understanding of the course material. The main criticism I have of her article is that it speaks of the experience of teaching writing over just one semester. She has good ideas, but they are not well tested. She also focuses primarily on the writing component of the course, as opposed to my synthesis of writing and content.

Meanwhile Dugan and Polanski explore teaching writing in CS from two perspectives, that of a CS professional and that of a writing program faculty member [4]. They chose to team-teach a software engineering course, with a strong writing component. Dugan brought his software engineering expertise and experience to the table. Polanski provided the writing components. Their research focuses on developing a writing taxonomy with three concentrations: writing for learning, writing for academic communication, and writing for industrial communication. In addition, they provide a variety of general-purpose advice when teaching writing. Perhaps the most valuable piece of their research is the large number of references provided for the various taxonomies. Perhaps a writing instructor would be familiar with sources such as these, but they are not the resources commonly circulated among Computer Science professionals.

At a recent CCSC:Northeastern conference a panel of Computer Science and English faculty met to discuss integrating writing across the Computer Science curriculum [5]. Faculty represent two universities and their writing across the curriculum initiatives. At Widener University engaged in a WI initiative much like our own at Seton Hill, requiring students to take a certain number of writing-enriched courses. Quinnipiac University initiated training for their faculty in teaching writing and collaborations with the English faculty members. The result of their program is the use of writing as a pedagogical component in all CS courses. In particular, writing is used to enhance student engagement and understanding. As writing is a valuable skill for all graduates, their model assists in achieving multiple important goals.

Kaczmarczyk reports a different approach taken at the University of Texas at Austin [3]. Their CS students take a technical writing class specifically focused on academic technical writing and industrial technical writing. A study was conducted which attempted to measure student motivation towards the course and material, mastery of academic writing skills, and self-efficacy towards academic writing. The study partially proved the goal of self-efficacy of students towards academic writing. The skill-based goals were more strongly supported. Students reported an increase in confidence in expressing themselves through writing and in mastery of the material. Students did not however report an increased motivation towards the class and the material, an additional goal of the course. The discussion of their survey results makes a questionable attempt at understanding the meanings behind the results of the study. Better evidence of the usefulness of the study may be found in their discussion on the changes that were made to the course. Results from the course after these changes would be interesting.

Garvey also discusses teaching Programming Languages as a writing intensive course [6]. Instead of focusing on one large project, Garvey requires a variety of assignments in his course. The primary focus of his paper is his argument that writing can be beneficial in many forms, such as summarizing material, documenting and peer reviews of code, reviews of the writing of others, speculative writing, arguing for a particular perspective, scientific research papers, interdisciplinary writing, and team-based writing. He also addresses the issues of requiring students to engage in writing as a process, and faculty evaluation of student writing.

7 Conclusion

Having taught Programming Languages as a WI course for three years, I can say I have definitely faced a learning
curve. Initially my writing assignments did not effectively express my vision of the assignment to the students. I clumsily led writing discussions. And evaluating student product required a significant amount of time. As my skills have grown, I have become both more effective and more efficient. I have reached my goals of adding a significant writing component to my Programming Languages course, and enhancing the educational value of the course. In addition my comfort level with teaching writing has increased; I no longer dread every other fall semester.

Like many tasks, the first 80% took me 20% of the time involved, and the last 20% will take 80% of the time. I continue to work to fine-tune my WI pedagogy to increase the critical thinking required of the students, to bridge the gap between the background level of the students, and to keep the writing assignment creative and interesting.

7.1 A final thought

Every year I learn something new. The past semester, I attempted to instruct the students to use the voice of authority in their writing by saying, “You are the master of your language.” The students apparently translated this statement into something like, “That means I can use ‘I think’ in every other sentence.” Not my intended result! Eventually I might figure out all of the little pitfalls. That is a big might. In the mean time, I will have an interesting time learning.

8 Acknowledgements

Special thanks to Dr. Dennis Jerz, Associate Professor of English and New Media Journalism at Seton Hill University and the students in CS250, Programming Languages, in Fall of 2011.

9 References


