Report 1 Study Leave 2012
Sociolinguistics in Programming: Background Material
TR-CSJR2-2012

1 Introduction

This report introduces the background and motivation for the research project "Sociolinguistics in Programming". Brief overviews of the relevant areas are provided, with emphasis on areas of relevance to this project. I also attempt to expand on connections between existing work in these areas and the work that I am proposing. Data collection and analysis is currently underway, and future reports will provide in-depth literature reviews and address methodologies and results.

2 Motivation

The initial motivation for this work was based on anecdotal evidence that different people programmed differently. While a long discussion can be held regarding the *process* of programming[1], what this research focuses on is the final product, or the result of the programming activity. How can two programs that have the same end result, or functionality, look so different? Are there aspects of these differences that are to be encouraged, (*i.e.* in students), or discouraged, and why? And finally, are there cultural or sociological reasons behind why a particular person's programming activity might result in a particular style being used? This report explores the reasons and background behind these questions and provides some discussion as to why it is interesting to ask them.

[1] The reader is directed to [1] for more information on work in this area.

3 Research Questions

Before delving further into this report it is important to put it into context. The project described by this report is attempting to answer the following research questions:

- do sociological factors such as gender, experience, or first language spoken play a role in how people write computer programs?
- if so what characteristics seem to be associated with these factors?
- if not, why not?

This is the initial phase of this work, and thus the questions may evolve in the future; however it is these questions that are the driving focus for the current project. In particular I am interested in whether there are differences in programs written by people of different genders, and if so how those differences might be characterised.

4 Sociolinguistics

Not being a linguist by training I am struggling with the concept of sociolinguistics. Thus for my own benefit and that of the reader of this report I provide some background on the areas of linguistics and sociolinguistics, and then discuss how ties to this project.

[2] defines a society as "any group of people who are drawn together for a certain purpose or purposes", and a language as "what the members of a particular society speak". These two definitions can easily be applied to the community of computer programmers; indeed the term "community of programmers" is already in common use, suggesting that within such community there is a society and thus a language in use. Of course when we refer to the language within a society we are generally referring to a natural language, and various studies such as [3] and [4] are looking at ways in which natural language is used in this society. This project, however, is examining how people use artificial languages; that is, programming languages. While it may not be obvious to a novice programmer, an experienced programmer is very well aware that when they are writing a computer program they are communicating not only with the computer, but also with other people who might need to read and understand their code. Thus my project is investigating the use of programming languages within the society, or community, of computer programmer.

There are many interesting technical terms that are used by linguists in reference to natural languages, but have interesting connotations when applied to artificial languages. For instance, Chomsky discussed the difference between *competence* and *performance* of a language, and [2] introduces these concepts in Chomsky's own words which I will paraphrase: performance is the use of the lanugage, while competence describes what speakers know about their language. From my own experience I would say I can perform the English language quite well; however my competence in the language is probably less than expert. In teaching programming, however, we can usually characterize a first or second-year programming student as having the opposite knowledge; that is, they generally have knowledge about the language, but are not good at performing it.

[2] goes on to discuss relationships between language and society. One such relationship is that social structure may either influence or determine linguistic structure and/or behaviour; for instance, *age-grading* refers to the phenomenon in which young children speak differently from older children, who in turn speak differently from adults. Another view, however, is that linguistic structure and/or behaviour may influence social structure; thus the language itself shapes the person; and yet another view is that both are taking place at the same time. The fourth relationship possibility is that there is no relationship between language and society. [2] cites Gumperz[2], in stating that "sociolinguistics is an attempt to find correlations between social structure and linguistic structure..."

Sociolinguists refer to the differences in how people use language in different ways. For instance, the English language may be said to have many *varieties*, such as Oxford English, Cockney English, Canadian English or legalese. The term variety seems to be hard to define; however I won't worry about it in this report as it is not relevant for this research. Similarly so for the term *dialect*, which generally is used to refer to one particular use of a given language, when such language is actually a group of related normal uses of that language [2]. This research isn't concerned with these distinctions because computer languages must be understood by compilers and interpreters – software that converts programs into machine language for the computer to executed. Thus for each programing language the dialect and variety is strictly controlled. That is, if a programmer tried to use a word that he or she had invented a new use for, the program that it was used in would simply not work and would have to be corrected before the computer could execute it.

[2]Gumperz, J. J. (1971), Language in Social Groups, Standford University Press p. 223

Style, however, is relevant to this work. I interpret style to mean some *correct use* of the language, but a use that is specific to the individual speaker, or a particular group of speakers. In natural language this could mean a preference for a particular dialect; in programming this could refer to how words are spaced, or how the program is organized, as long as the program will execute.

Another interesting topic is that of code-selection, and code-switching. If a person understands and is fluent in many programming languages, and how might they employ code-selection and code-switching in their development of a solution? Would this wider choice of solutions, or style of solution maybe, be reflected in their final program? This is beyond the scope of this work, however, and so the reader is directed to [2] ch. 4 for further explanation of these terms from the sociolinguistics point of view.

As this research is concerned with language and culture, I touch on some of these aspects here. One area of interest is the Whorfian hypothesis, in which Benjamin Lee Whorf suggested that the structure of a language determines the way in which the speakers of that language view the world [2]. Can the same hold true in programming? As will be discussed in Section 8.4.1 there are a variety of programming languages, some of which exhibit significantly different characteristics. Problem-solving can be approached in very different ways, depending on the language you are using to write the software solution. Thus would a programmer trained to write programs in an imperative language approach problems differently than a programmer trained in functional languages? Although it may be difficult to collect data on this, this is one area of interest encompassed by this project.

5 Gender and Language

Men and women use language differently; this is argued in a variety of different ways by the authors references in this section. Research into language is now approached somewhat differently, however, and rather than implying that because one is a man/woman one must speak and/or write in a particular way, many researchers in this area are now leaning more towards critical discourse analysis that is, in the words of Talbot, "committed to examining the way language contributes to social reproduction and social change." [5] (p. 149). Such analysese examine how people use language as they are "doing" gender. For example, a woman who works in an engineering field outside of the home might, in the course of her engineering work, use language in such a way that lends itself to fitting in with her primarily male

colleagues and to asserting domination in her field; this would generally be labeled as "masculine" characteristics. Conversely, that same woman might then also be a mother, using nurturing and encouraging language, language that might be labeled as having "feminine" characteristics.

In addition, Kendall states that "Women and men actively choose ways of framing to accomplish specific ends within particular interaction. These choices are drawn, in part, from sociocultural norms for how women and men are expected to accomplish such actions..." [6] (p. 82), and it is this frame of mind that I wish to maintain as I moved forward through this investigation.

There is huge variety in how researchers have approached this area, ranging from qualitative to quantitative works; sociological approaches, feminist approaches, skeptical approaches and combinations of these and many others. Future works stemming from this project will no doubt address this area in more depth; however in this report I attempt to extract from the literature a smattering of relevant works to provide support and direction for this work. I am, of course, aware that at this point I have only just scratched the surface of this work, and works such as [7], [8], [9], and [10] are recommended to the reader as a start in this area.

5.1 Female and Male: intercultural differences

"The Way Women Write" by Mary Hiatt [11] presents one of the first investigations into whether women and men used different styles in their writing. Hiatt asked a number of questions including:

> If there exist a masculine style and a feminine style, in what ways are they distinct? How do they differ? Do women write less well than men? ... And how is a judgment to be made?

(p. 2). As she goes on to note, critics' "judgment" of style tended to reflect gender regardless of the actual writing, but based almost solely on the author's name and thus perceived gender. Hiatt's work addresses a study based on a selection of 100 books, and utilised 2000 words from each book which were painstakingly typed onto punchcards in order that computer analysis could be performed – possibly one of the earliest examples of corpus techniques for this purpose? I think it is worth noting that the researcher taught herself SNOBOL in order to develop the software for this analysis. Her findings were indeed that women write differently than men in a variety of different ways. Some criticism might be given of this work, in that by demonstrating these differences Hiatt is actually giving validity to the male

critics who posited these differences in the first place; however it is an interesting place to start, and the study provides a first, as far as I am aware, in this corpus-based approach to analysis of style from a sociolinguistics point of view.

In other work in this vein, Mulvaney writes that "Communicative practices not only reflect notions about gender, but they also create cultural concepts of gender" [12]. This is particularly relevant in this work, as we investigate a restrictive form of communication, in the form of writing a computer program, but also as we consider the gender biases that surround computers and technology. Mulvaney considers communication between genders an "intercultural experience" (from which the title of this subsection was taken), implying that the differing cultures and experiences of a man and a woman cause them to interpret communications differently. In the context of our work, would this mean a man might write a more terse, concise program that does not consider any future human reader, and a woman might be more "wordy" and include more explanation as she considers how another individual might struggle with understanding her meaning? And would that future individual find the extra explanation to be useful, or to be clutter, and will this depend on their experiences and possibly, gender-shaped beliefs? Mulvaney also goes on to state the following: "Language also reflects differences in social status between genders. Research on gender and language reveals that female language strategies invariably emulate the subordinate, nonaggressive role of women in Western society." Are these female language strategies apparent in the use of programming languages? Can we learn from Mulvaney's analysis and produce styles of communication, including that of writing programming languages, that might be easier to read for all individuals?

Some articles doubt that there are significant differences in language use; those like [13] present evidence that there are little true differences in the abilities and behaviours among genders, However recent studies such as [14], [15] and [16] that have quantitatively examined gender differences in language use have found significant differences. Of particular interest to me is a mention in one study of a connection to testosterone and language: "Overall, testosterone had the effect of suppressing the participants use of non-I pronouns. That is, as testosterone levels dropped in the weeks after the hormone injections, the participants began making more references to other humans. Contrary to stereotypes about the subjective experience of energy, positive affect, heightened sexuality, and aggression thought to be related to this hormone, no consistent mood or other linguistic correlates of testosterone emerged. One function of testosterone, then, may be to steer

peoples interests away from other people as social beings." [16]

5.2 CMD

Another area of interest in relation to this project is that of computer-mediated discussion/communication, often abbreviated as CMD or CMC. Since the author's anonymity can be, if the author wishes, preserved, identification of gender is left solely to the characteristics of their writing. In studies of CMD researchers such as Coates hav found that linguistic characteristics signalling gender are similar to those described in face-to-face interaction [17]. However Herring clarifies this observation as follows: "There is an overall tendency for some of these behaviors to correlate more with female CMD users, and for others to correlate more with males. This does not mean that each and every female and male manifests the behaviors; exceptions to the tendencies can readily be found. It does mean, however, that gender predicts certain online behaviours with greater than chance frequency when considered over aggregate populations of users, controlling for variables such as age, topic and the synchronicity of the medium" [18].

Also as described by Herring, politeness is another clue used to predict gender in CMD. She reports on studies in which women are more likely to be bothered by violations of acceptable usage guidelines to ensure polite exchanges, and on other studies showing that computer-based chat groups run by women are more likely to have such guidelines and of a stricter nature than groups run by men [18].

5.3 Politeness

Politeness is an area that can be further investigated, although the definition of politeness in a computer program may be difficult to agree upon. Is it polite to use long words that explain the use of the program? Or are shorter words more polite, allowing the reader to parse the program more efficiently and effectively. Similarly for comments, or non-computer-parsed explanations; are more, or less a better way to program? Clearly this will be open to interpretation and individual opinion, and research on the positive and negative aspects of each of these, in terms of understanding and reading a program would be useful in supplying a base-line from which to begin. Some work from linguistics addresses this in a verbal context. We have all participated in awkward conversations, and similarly had an "easy" conversation with someone. Why the differences? Part of the differences we experience when conversing with different people might be attributed to

what [2] (quoting Grice [3]) calls the cooperative principle: "Make your conversational contribution such as is required, at the stage at which it occurs, by the accepted purpose or direction of the talk exchange in which you are engaged." There are then four maxims that follow from this: quantity; contribute as much information as is required; quality; don't lie, or make things up; relation; only provide what is relevant; and manner; avoid obscurity and ambiguity, and provide information in a brief and organized way. Moreover Grice points out that speakers don't always follow these maxims, and in doing so may actually imply something different from what is being said. In a similar category falls the concept of 'face'; that is, we accept what others are presenting to us as being the truth, and that they are presenting to us a true image of themselves. Conversation also involves 'face-work'; presenting faces to one another, and protecting both our own and the other's face [2] (citing Goffman[4]). Thus participating in a conversation requires effort on both parties, but that effort can be lessoned by appropriate cues, signals, and actions by each participant. Are there lessons to be learned here for our computer programs?

5.4 Power

Another topic that might be addressed is connections between language and power. Indeed, the well-known McConnell and Eckert state that "...power has been the engine driving most research on language and gender, motivated partly by the desire to understand male dominance and partly by the desire to dismantle it..." [19] (p. 474). In the context of this work we rule out choices among languages, *i.e.* choices that might be made in a multi-lingual community; however the way in which one particular language is used can still communicate power (or the lack of it). This report does not address this area, although the reader might be suggested to begin with resources such as the following: [20], [21] or [22].

5.5 Discourse

This subsection seems somewhat redundant to me, as all investigation of language involves an investigation of discourse, discourse being any verbal or written form of communication. However many researchers in a variety

[3] Grice, H. P., 1975. Logic and Conversation, In Cole, P. and Morgan, J. L. (eds), 1975. Syntax and Semantics. Volume 3, Speech Acts. New York, Academic Press.

[4] Goffman, E., 1955. On Facework: An Analysis of Ritual Elements in Social Interaction. Psychiatry, 18:213–231. In Laver, J. and Hutcheson, S. (eds), 1972. Communication in Face to Face Interaction. Harmondsworth, England: Penguin Books

of areas have written on gender and discourse. The in-depth investigation of this area falls within the scope of an up-coming report related to this project; however for now we direct the reader to texts such as [23], [24] and [8].

6 Sociology, Anthropology, and Culture

This report makes an effort to pigeon-hole each related topic, and discuss it separately from the big picture. Of course, this just isn't possible. Eckert and McConnell-Ginert's article "Communities of practice: Where language, gender, and power all live" supports this, in a sense, commonsense idea [19]. However in the context of this work, this means that discussing gender and language is somewhat meaningless unless one also considers the context and community: that of computer programmers, and computer programming. Och's work on physicists [25] is an interesting approach describing how physicists use language to tie in visualizations and personify the scientific constructs, and an examination of the process and discussions surrounding programming might be equally interesting and illuminating. I feel, however, that this is beyond the primary scope of this project. What is not beyond the scope is consideration of how the culture of computers and computer programming might affect the members of this community.

Many studies have examined connections between gender and computer use and computer beliefs. For instance Sigurdsson's study found that experience, rather than sex differences made a bigger difference in attitudes towards computers. However he also reported the following: "...females were significantly less positive toward computers than males. Females have less experience of computer languages and use computers less often for programming than males (although not significant). Sex differences on the personality variables appeared to be much more significant. Females were more emotional, dependent, self-critical and neurotic than males and they were also more extraverted although not significantly." [26]. Other studies such as [27] made similar findings.

Whitley in his meta-analysis of studies finds that the overall gender differences in beliefs about computers, affects, self-efficacy, and behaviour towards them are small; however he finds a large difference in high-school sex-role stereotyping. One can interpret this to mean that according to the studies he examined, there is little difference in how children, adults, or teens approach computers or believe in their abilities about computers, *except* when it comes to teens, who believe that there is a large difference in

who should be working with computers. In Whitley's own words, "Boys and men, compared with girls and women, saw computers are more appropriate to themselves, saw themselves as more competent on computer-related tasks, and reported more positive affect toward computers. " [28]. We see the results of this in universities today; fewer women than men in directly computer-related fields such as computer science and engineering, and higher drop-out rates for women than for men in these fields [29]. Klawe and Leveson have pointed out that despite the advent of the internet, which purportedly has provided equal opportunities to people regardless of gender, fewer than 35% of IT professionals are women, and the number of female college students majoring in computer-related fields has actually declined, rather than increased as the popularity of the internet grew through the 1980s and 90s [30].

[31] provides a more recent analysis (2005) as well as an overview of the plethora of studies; their findings differ in that they report females as having higher confidence and attitudes towards the importance of computers, but males as having better skills. Verkiri and Chronaki also present a study of gender and computer attitudes, with perceptions of social support thrown in, but statements such as "...even though during the 70s and the 80s one could discern noticeable patterns of girls under-achieving in specific areas such as mathematics and science, today the media sound bites reverse the story and construct boys as having difficulties" [32] suggest that cultural differences are at work. In North America the media is certainly not "reversing the story", as evidenced by Sanders' 2006 report [33]. Having said that, Vekiri and Chronaki's study focuses on the "gendered expectations" surrounding computers, and identifies a need for educational and social "interventions".

Walsh, Hickey and Duffy report also on an interesting finding; that male or female labeling of the characters in math problems did not account for gender differences in performance, but prior beliefs that gender differences existed (for a particular test) did result in women scoring lower than men [34]. Could this imply that women, believing that computers are not appropriate choices for them as careers, may self-fulfil this prophecy by performing worse with computers? On the other hand, what does "worse performance" mean, when a novice programmer is faced with a brand new language and they have no concept of good style or effective habits? One might assume that their "natural" use of language would creep into their approach.

The evidence is varied, but the cultural impression seems to be that computers are something that is better suited to males, whether for work (*e.g.* career-choices) or pleasure (*e.g.* playing computer games). We see

evidence of this in the small percentages of women in our computer science classrooms today. The question for this project is how might this impact how people write computer programs, or does it?

Finally, it is worth mentioning Trechter's chapter in Holmes and Meyerhoff's Handbook of Language and Gender. Trechter discusses how gender is "done" within various contexts of ethnicity [35]. This is relevant to this work, in that Trechter discusses how behaviours change in order to for an individual to express a gender, or to encourage a sense of "belongingness" to a particular group. This may be evident in this work, as people may feel the need to "fit in" to the perceived view of a programmer, but also are doing gender at the same time.

7 Psychology of Programming

Weinberg [36] first coined this term and literally wrote the book on the psychology of programming. He was the first to connect the dots between the social aspects of writing computer programs and the end result. While this book was written in 1971, in an era where one might argue that programming was just beginning, Weinberg provides evidence and anecdotes to support the ideas that programming was (and still is) an activity affected by various factors, not the least of which is personality. In fact, Weiberg writes an entire chapter on personality factors, in which he states "isn't there some way we can select those people whoe personalities suit them for programming"? However some of the important factors that he selects include neatness and ability to deal with stress, both of which I think are no longer relevant. Or rather, one might say that these are important factors in nearly all careers, not only programming! However he then goes on to mention assertiveness and humility, and interestingly enough found that what he calls "humble" programmers perform better in batch environments, where they were careful to never make the same errors, and methodically checked their changes to ensure that every run after the first few was free of syntax errors. Conversely he argues that the "assertive" programmers performed better in an online system, where hasty changes were made to fix things, which invariable led to additional problems which would require additional fixes, but that this trait would leave the "humble" programmer "gaping at the terminal". This may not hold true in today's world, since batch processing is not as common as it used to be; still, there may be a grain of truth linking certain style characteristics to different types of projects and/or programming-related tasks. In a following chapter Weinberg discusses some principles for programming

language design, which might be extrapolated to programming language *use*. These principles are described below. UNIFORMITY: a language that contains many deviations from a given rule (or set of rules) will violate the principle of uniformity and will be more difficult to learn, and more difficult to use without error. COMPACTNESS: Since the human mind has inherent limitations in capacity, a short program (in general) will be more easily comprehended than a long one. Even the addition of non-executable text tended (in his studies) to increase the difficulty in reading a program, even when the intent of the additions was exactly the opposite! Compactness is not merely counting the number of characters, however; since a word such as "BIG" can be handled (by the human brain) just as easily as a single character e.g. 'A'. Weinberg relates this idea of compactness to chunking; that is, a recoding of longer chunks of information to a smaller piece that one can encompass in its entirety. Abbreviations may help this, e.g. rewriting Procedure as Proc to compress the section of information that is relevant. LOCALITY and LINEARITY: Locality refers to the property that obtains when all relevant parts of a program are found in the same place, while linearity refers to the idea that when events occur in a predictable sequence, e.g. the notes in a tune, they are easier to recall and comprehend as the appearance of each item "triggers" the next. TRADITION AND INNOVATION: what comes 'naturally' to the programmer, (and to the reader?) making a programming language in some sense consonant with the other languages a programmer knows.

Weinberg's seminal work has spawned an entire area of research and study including books such as [37]. A great deal of work in this area is examining topics such as how people explain, and in turn understand, existing code [38] and researchers such as Burnett are looking at gender differences in how people interact with computer (human computer interaction, or HCI) [39]. This is, of course, a small selection of the entire area, but highlights some areas that may be of relevance to this project.

8 Programming: Good and Bad

People discussing links between computers and linguistics are generally referring to the area of computational linguistics. Computational linguistics examines natural language, often with the goal of developing software that will allow computers to "understand" statements written in natural languages. In this research I am going the other way; we have statements written in an artificial language, designed for a computer to "understand",

but what would a human say about these statements? This section of the report talks about programming and how people can make use of programming languages in different ways.

Looking at the literature on programming style, there is clearly an understanding that some styles are "good" while others are "bad". This research grew out of an observation that my programs that I would write looked very different than those of one of my colleagues, despite both being functionally correct. Thus my initial thought was simply to compare styles to see whether people might use programming languages in ways that were either more or less "readable". That is, programming languages are unlike natural languages in that they are artificially created, and the rules behind the structure of a sentence simply cannot be broken. If those rules are broken, the program will not work. Programmers, or people creating a text using some programming language are always aware that the program they create will be compiled by a computer. The process of creating a computer program involves a person writing the program, a computer compiling the program (or more accurately, by a computer executing the instructions contained in another program), and finally the computer can then execute (carry out) that set of machine language instructions to achieve the desired functionality. Thus one of the programmer's goals (and sometimes their only goal) is to write a program that can be compiled to machine language and successfully carry out whatever functionality was originally desired. This is not necessarily a good thing, however; with the growing complexity of computer software it is becoming essential to also write programs with a human audience in mind. I will elaborate on this further on.

Because all programs are designed to be compiled and then executed (carried out) by a computer, computer languages of necessity are designed to be context-free. This means there is one, and only one way to interpret statements written in computer languages[5]. However, within those rules there is some flexibility, most notably in the use of *comments* and in the naming of *identifiers*.

[5]Some comment on context-free grammars is probably appropriate here, in that they play a large part in both computer science and linguistics. Having said that, and as I approach page 20 of this report I feel that an in-depth discussion would not add significant understanding to the reader, and thus I suggest a textbook such as [40] to those who are interested in this area.

8.1 Comments

Comments are sections of a program that are not designed for the computer; they are, in fact, ignored when a program is compiled, and they are generally written in a natural language. There exist some conventions for the use of comments, particular if automatic documentation generation (e.g. Doxygen [41]) is going to be used, but there is still the flexibility of how to explain things within the format required by such tools.

8.2 Identifiers

Identifiers are the words used in a computer program. There are generally rules around what constitues a "legal" identifier, such as restrictions as to what characters may be used and where they may appear. Some identifiers have pre-existing definitions within the language; these are referred to as *reserved words* and (in most imperative languages[6]) include words such as "if", "else", "while", "for". When a programmer is defining a block of statements (a function or procedure) one generally labels this section with some identifier; as long as the identifier is not a reserved word in the language and follows the rules for creating an identifier, the sky is the limit. Thus the identifier for a section of the program that counts the items in a list could be CountItems, count_items, doCountItems, mycountingprocedures, or x. Of course, these are just a few of the almost limitless possibilities! Identifiers are also used for naming variables, which can be briefly explained as locations in the computer's memory where data can be stored for use by the program. The same rules are followed in this case.

8.3 Other

Despite the context-free nature of a programming language, it is still possible to express the same thing in many ways. For example, to determine if the value stored in variable X is less than 18 one might write any one of the following:

```
if (X < 18) ...

if (18 > X) ...

if not (X >= 18) ...
```

[6] see section 8.4.1 for an explanation of imperative languages.

These all have the same meaning, but even a non-programmer will probably see that one of these options is much clearer than the other two.

Organization of statements is also flexible, to a certain extent, and can make a big difference to how a program looks. For instance, it is common practice to group sections of code according to what function they are carrying out, and to declare all variables used within these groups at the start of each group. However this is not required, and variables could be declared anywhere within the block as long as they are declared before one attempts to store data in them.

White space is also a big part of the look and feel of a program. Like comments, the compiler ignores all white space in a program. A human reader, though, can easily appreciate its use when comparing the two examples below:

```
#include <iostream>
using namespace std;
void main()
{
  cout << "Hello World!" << endl;
  cout << "Welcome to C++ Programming" << endl;
}
```

```
#include<iostream> using namespace std;void main(){cout<<"Hello World!"<<endl;
cout<<"Welcome to C++ Programming"<<endl;}
```

Even for an experienced programmer it is more difficult to parse the statements and separate out identifiers and tokens in the second example.

Having defined some of the ways in which the use of programming languages can vary, I next address some of the research into the quality of software and computer programs.

8.4 Quality

The quality of software is both hard to measure, and very important; current research articles (e.g. [4]) as well as popular Software Engineering textbooks (e.g. [42]) provide many convincing (and sometimes scary) examples of how problems with software have affected people's lives as well as the bottom line. There is an argument that the quality of a piece of software can be measured simply by how many bugs have been found in it, but there are many problems with this. First of all, just because a bug hasn't been found *yet* doesn't mean it doesn't exist. It is not possible to prove that a program

is bug-free! Secondly, software is becoming so complex and having so much invested in it that we can't afford to throw it away when the next change is needed. We have to continue to modify, maintain, and grow our software systems, and that means people have to read, understand, and build on the existing programs. Thus one can make a good argument that quality of software could be measured by how easy a programmer finds it to work with a particular program.

The following sections will explore some of the relevant research in this area, with the goal of setting the stage for the directions that my project is taking.

8.4.1 Types of Programming Languages

[43] discusses an interesting idea; that is, that a particular programming language, or even type of programming language, is inherently more readable than another language or type. The author compares two programming languages, Miranda and Pascal. Miranda is a functional programming language, while Pascal is imperative. The underlying model is quite different, making it difficult, sometimes, for a programmer to move from one type of language to another. As described by the author,

> The imperative model incorporates the Von Neuman machine characteristics in the notions of assignment, state and effect. A characteristic of this language class is the explicit flow of control, *e.g.* sequencing, selection and repetition. In assignments, the value of memory places denoted with variables in changed during program execution. This model of operation by change of state and by alteration of variable values is also named 'computation by effect'.

Thus a programmer writing in Pascal must consider the series of steps which must be carried out in order to complete the desired task, and as well where final and intermediate values required for the computation will be stored. The functional model, in comparison, is quite different:

> The functional model model is characterized by 'computation by value'. Functions return values and have no side-effects, and expressions represent values. There is no notion of updatable memory accessible by instruction. The program consists of a script with a number of mathematical-like definitions and an expression that must be evaluated. Functions can be passed as

arguments to other functions and can be the result of a function application (higher-order functions).

The author argues that this model is more closely related to mathematical activities[7]. Because of this relationship there is an argument that the functional style leads to more readable, and less fault-prone, programs. In addition the author also argues that learning programming in a functional style has advantages to learning in an imperative style. For example, in order to compute the factorial of a number, in Miranda the code would read

```
fac n   = product [1..n]
```

In an imperative language such as Pascal the code would be rather longer, something like

```
function fact(n: integer): longint;
begin
    if (n = 0) then
        fact := 1;
    else
        fact := n * fact(n - 1);
end;
```

In the imperative language (Pascal) the programmer has to be aware that memory storage is needed (a variable, in this case n), and as well has to break the problem of computing a factorial into individual steps that can be computed in a sequential way. That is, one has to see that a value $1 \times 2 \times 3 \times \ldots \times n$ can be computed by finding $n \times n - 1$, then multiplying that by $n - 2$, then multiplying that by $n - 3$ all the way down to where we are multiplying by 1. Of course, there are other ways to write this Pascal function, but the same ability to break the problem down is required. In Miranda we can see that the code is much simpler. However as more complex tasks are required the syntax may not be as simple; defining a list containing powers of 2 up to a value of n can be done like this:

```
powers_of_2 = [ n | n <- 1, 2*n .. ]
```

[7]This could lead to an interesting discussion on today's culture, and whether, with the current attitudes towards mathematics, a functional language would be more or less difficult to introduce to novices, either in general or to particular groups *e.g.* boys or girls, and whether cultural attitudes towards mathematics would affect how quickly people learn a functional language in contrast to an imperative language.

but the programmer (and reader) has to understand the meaning of all the symbols on the right of the equals sign. Now for a mathematician, this looks very much like a mathematical formula, but for someone without that background this would not necessarily be clear. I should note, in the sake of fairness, that defining a similar list in Pascal would require a great deal more lines of code; something like this:[8]

```
procedure squares(n: integer; var squaresList :Array[0..n] of longint);
begin
    squaresList[0]=1;
    for count:=1 to n do
       squaresList[count] = 2*squaresList[count-1];
end;
```

The experiments reported on in [43] computed known static syntactic complexity measures which the author equated with the idea of "readability". We discuss readability further in section 8.5. The author then compared the values computed for these metrics with evaluations of readability by lecturers in Computer Science, and found that there was a significant correlation between the lecturers' assessments of the readability of the Pascal programs and the values of the complexity metrics; however the correlation was much lower for the Miranda programs. The author explains this as possibly being due to a lower standard of readability for Miranda programs. I personally suspect that in this case the imperative language, being more "English-like" is easier to evaluate for readability, while it might be more appropriate to ask mathematicians to evaluate the readability of a Miranda program. In summary this paper provides a nice discussion on imperative versus functional programming languages, and attempts to compare them; however for my work the primary use of this paper is to provide some illumination for my choice of languages in the study being proposed.

8.5 Readability

As indicated in the previous section, the idea of "readability" seems to be of importance. The authors of [44] define this as follows:

> a human judgment of how easy a text is to understand.

[8] Yes, I know this won't actually compile in Pascal because the size of arrays must be known at compile time; however the equivalent block would work just fine in other imperative langauges such as C. The point was just to show the difference between the code in an imperative language versus a functional language, and the code samples in this report are to be treated as pseudocode.

Why is it so important? As discussed earlier, computer programs are no longer written solely for the computer, but equally so for the human reader. [45] indicates that %70 of the total lifecycle of a piece of software is solely maintainance; this means having people read, modify, and correct existing programs. The same paper talks extensively about ways to reduce defects in software, and many of the techniques they discuss involve having other people review one's code. These arguments all point towards how important it is to write a program that is easy for people to read and understand.

[44] reports on the development of a metric that is used (in their case on programs written in Java) to measure readability. As in the previously discussed work, they then automate the measurement of the metric and compare it to human evaluations. The correlation between the human evaluations and the automated metric is quite high, although they note some interesting results. The metric they discuss is composed of a variety of measures such as lengths of variable names and lines used in the program (maximum and average); numbers of keywords (reserved words), numbers, and identifiers (max and average); and average number of comments, and average numbers of other tokens commonly used such as periods, commas, spaces, and parenthesese. Other measures taken included branches (use of the word if) and loops (use of for or while). In order to build their metric they correlated each of these measures individually with their human evaluations, and found that factors like average line length and number of identifers were very important to readability, but that average identifier length, and numbers of branches, loops and comparison operators were not. This suggests that readability for programs is connected with being able to quickly "grab" a portion of the code with they eye, and allow the brain to consider it as a whole. That is, if a line is too long, the person reading it cannot take in the line in its entirety, and has to do extra work in order to process it. Similarly if too many unfamiliar words (*i.e.* identifiers) are used then more work in figuring how to understand them must be carried out in order to understand the code. This ties into ideas first introduced by Weinberg [36], who as far back as 1971 talked about principles for programming language design that included ideas such as uniformity, compactness, locality and linearity. Uniformity referred to ensuring consistency with the rules of the language, while compactness encompassed the idea of breaking things down into understandable chunks. Locality and linearity refer to keeping similar ideas together and to arranging things in a logical order. While Weinberg was suggesting that an entire language should be designed along these principles, in practice this hasn't really happened. However many online experts [46, 47, 48, 49], as well as programming reference books such

as [50] list principles similar to these when suggesting what makes good programming style[9].

8.6 Code Smells

I loved the title of this blog [48] so much that I am borrowing it for this report. The blog discusses things that the author has identified as warning signals that something in the code might be wrong, or if it isn't wrong now, it could go wrong in the future. The author uses these as signs that there is something in the code that should be fixed. While there is no experimental evidence presented that these are examples of "bad" programming style, the author does give reasons why each of the items listed are a bad idea. A selection of these are listed below:

- Comments: There's a fine line between comments that illuminate and comments that obscure. Are the comments necessary? Do they explain "why" and not "what"? Can you refactor the code so the comments aren't required? And remember, you're writing comments for people, not machines.
- Long Methods: All other things being equal, a shorter method is easier to read, easier to understand, and easier to troubleshoot.
- Long Parameter Lists: The more parameters a method has, the more complex it is.
- Duplicated code: Duplicated code is the bane of software development. Stamp out duplication whenever possible. Be on the lookout for more subtle cases of near-duplication, too.
- Conditional Complexity: Watch out for large conditional logic blocks, particularly blocks that tend to grow larger or change significantly over time.
- Large Classes: Large classes, like long methods, are difficult to read, underst and, and troubleshoot.
- Uncommunicative Names: Does the name of the method succinctly describe what that method does? Could you read the method's name

[9] It is worth noting that when going to Steve McConnell's blog [46], the first two entries are on technical debt, a term referring to really poorly written code that no one has had time to go back and fix.

to another developer and have them explain to you what it does? If not, rename it or rewrite it.

- Inconsistent Names: Pick a set of standard terminology and stick to it throughout your methods. For example, if you have Open(), you should probably have Close().

As an experienced programmer these seem reasonable guides, and reflect lessons that I have learned in my career (usually the hard way). Thus I would argue that if we turn this around, and identify (for instance) programmers who consistently use good names, reasonably-sized classes, short parameter lists and so on, we would find programmers who write "better" quality and easier to read code.

8.6.1 Words (Identifiers)

Clearly many of these "code smells" or the lack thereof can be difficult to automatically identify, thus requiring a human to evaluate the code. This would be, in part, the job of a teammate or a marker, should the work be for a course. Many of us would agree that An automatic evaluation of code would be a useful thing in marking, for self-evaluation, and in industry. Thus I next consider what, exactly, can be evaluated in a useful (*i.e.* automatic) way. Class lengths, numbers of parameters, and complexity of conditionals can all be identified; indeed the metric considered in [44] does many of these. Duplication could also be automatically identified. What may be more difficult is examination of the meaning of names. This seems to be a problematic area, with work such as [51] looking at how natural language analysis can be applied to source code to try and provide clues as to how the program works.

Splotsky discusses the use of identifiers in code, but from a different approach: in [47] he talks about how to choose identifiers that will tell you if your code is incorrect. This isn't a novel idea, as the basic idea was introduced in 1976 in a version called Hungarian Notation [52]. The idea is to name your identifiers in such a way that you can tell what is supposed to be stored in them. Thus rwMax is a variable that stores the maximum row size. Then if you you see a statement like rwMax= -5 a little alarm should go off in your head, even if you've never seen the program before and have no idea what it does. [47] gives some additional examples of how to ensure a particularly nasty web security vulnerability could be avoided by using this type of naming convention.

9 Conclusion

This report has provided an exploration of the fields related to the proposed project. To remind the reader, the research questions I propose to investigate are

- do sociological factors such as gender, experience, or first language spoken play a role in how people write computer programs?
- if so what characteristics seem to be associated with these factors?
- if not, why not?

As described in this work there are a variety of reasons for asking these questions, and similarly a variety of ways to attempt to answer them. I have found as I have written this report that it has served a number of purposes: it has illuminated how much more research and background is needed in the various areas related to this project; it has raised many, many more related questions that could be investigated; and it has demonstrated how truly multidisciplinary this project must be. Most importantly, however, the writing of this report has enhanced my knowledge of the various areas that can be brought in as investigative tools for this work.

10 Appendix: Questions and projects to propose

- code-selection and code-switching in programming languages: an investigation.
- is there gender bias in how we teach programmming, e.g. in the language used in the teaching process? How does this affect the end results?
- doing gender through programming
- do conversational differences show up in programming style? E.g. if an individual is more or less polite how might that affect their programs?
- can research in linguistics be applied to more effective teaching of programming languages?
- does the first language a person speaks (native language) affect how they program? similarly for the the first language they learn to program in?

- do people who are fluent in a programming language use natural language differently?
- how can we (and do we need to) redefine Chomskys notions of competence and performance, in terms of programming languages?
- what is the effect of age-grading on programming language use? (age-grading in programming: does it exist, and how is it characterized?)
- can we, and how should we proceed to, build a corpus of commonly-accepted usages of programming languages? what would be the advantage of such?

References

[1] Paul Ralph. Software design science research program, 2012. web page for research program, http://paulralph.name/research/software_design/, accessed March 2012.

[2] Ronald Wardhaugh. *An Introduction to Sociolinguistics*. Blackwell Publishers, Oxford, UK, 1992.

[3] Francois Taiani, Paul Grace, Geoff Coulson, and Gordon Blair. Past and future of reflective middleware: Towards a corpus-based impact analysis. In *Proceedings of the 7th workshop on Reflective and adaptive middleware (ARM)*, pages 41–46. ACM, 2008.

[4] Nicolas Bettenburg and Ahmed E. Hassan. Studying the impact of social structures on software quality. In *Proceedings of the 17th IEEE International Conference on Program Comprehension (ICPC '10)*, pages 124–133. IEEE, 2010.

[5] Mary M. Talbott. *Language and Gender*. Polity Press, 1998.

[6] S. Kendall and D. Tannen. Gender and language in the workplace. In R. Wodak, editor, *Gender and Discourse*. Sage, London, 1997.

[7] Holmes and Myercroft. *Handbook of Gender and Language*. Wiley-Blackwell, 2003.

[8] A. Weatherall. *Gender, Language and Discourse*. Taylor & Francis, 2002.

[9] P. Eckert and S. McConnell-Ginet. *Language and Gender*. University Press, Cambridge, UK, 2003.

[10] Mary M. Talbott. *Language and Gender*. Polity Press, 2nd edition, 2010.

[11] Mary Hiatt. *The Way Women Write*. Teachers College Press, New York, NY, 1977.

[12] B. M. Mulvaney. Gender differences in communication: An intercultural experience. downloaded May 2012.

[13] J. S. Hyde. The gender similarities hypothesis. *American Psychologist*, 60(6):581–592, 2005.

[14] M. L. Newman, J. W. Pennebaker, D. S. Berry, and J. M. Richards. Lying words: Predicting deception from linguistic style. *Personality and Social Psychology Bulletin*, 29(5):665–675, May 2003.

[15] M. L. Newman, C. J. Groom, L. D. Handelman, and J. W. Pennebaker. Gender differences in language use: An analysis of 14,000 text samples. *Discourse Processes*, 45:211–236, 2008.

[16] Cindy Chung and James Pennebaker. The psychological functions of function words. In K. Fiedler, editor, *Social Communcation*, pages 343–359. Psychology Press, New York, 2007. chapter 12.

[17] J. Coates. *Women, Men and Language*. Longman, London, 2nd edition, 1993.

[18] S. Herring. Gender and power in online communication. In J. Holmes and M. Meyerhoff, editors, *Handbook of Language and Gender*. Blackwell, 2003.

[19] P. Eckert and S. McConnell-Ginet. Communities of practice: Where language, gender, and power all live. In K. Hall, M. Bucholtz, and B. Moonwomon, editors, *Locating Power, Proceedings of the 1992 Berkeley Women and Language Conference*, pages 89–99, Berkeley, 1992. Berkeley Women and Language Group.

[20] N. Fairclough. *Language and power*. Longman, 2nd edition, 2001.

[21] P. Simpson and A. Mayr. *Language and Power: A Resource Book for Students*. Taylor & Francis, 2010.

[22] P. Kunsmann. Gender, status and power in discourse behavior of men and women. *Linguistik online*, 5, Jan. 2000.

[23] R. Wodak, editor. *Gender and Discourse*. Sage, London, 1997. accessed May 2012 online at http://lib.myilibrary.com.ezproxy.lancs.ac.uk?ID=255966.

[24] L. Litosseliti and J. Sunderland. *Gender identity and discourse analysis*. John Benjamins, Philadelphia, PA, 2002.

[25] E. Ochs, S. Jacoby, and P. Gonzales. Interpretive journeys: How physicists talk and travel through graphic space. *Configurations*, 1:151–171, 1994.

[26] J. F. Sigurdsson. Computer experience, attitudes toward computers and personality characteristics in psychology undergraduates. *Personality and Individual Differences*, 12(6):617–624, 1991.

[27] T. Busch. Gender differences in self-efficacy and attitudes toward computers. *Journal of Educational Computing Research*, 12:147–158, 1995.

[28] B. J. Whitley Jr. Gender differences in computer-related attitudes and behavior: A meta-analysis. *Computers in Human Behavior*, 13(1):1–22, 1997.

[29] J. E. Rice. Being a woman in computer science in alberta. In G. Bonifacio, editor, *WGST 1000 Workbook: Gender Vista*. University of Lethbridge Print Services, 2012. published as part of course pack for Fall 2012.

[30] M. Klawe and N. Leveson. Women in computing: where are we now? *Communications of the ACM*, 38(1):29–35, January 1995.

[31] T. Faekah and T. Ariffin. Gender differences in computer attitudes and skills. *Jurnal Pendidikan Malaysia*, 30:75–91, 2005. downloaded from http://journalarticle.ukm.my/152/ May 2012.

[32] I. Vekiri and A. Chronaki. Gender issues in technology use: Perceived social support, computer self-efficacy and value beliefs, and computer use beyond school. *Computers and Education*, 51:1392–1404, 2008.

[33] J. Sanders. Gender and technology. In C. Skelton, B. Francis, and L. Smulyan, editors, *Handbook of Gender and Education*. Sage, London, UK, 2006.

[34] M. Walsh, C. Hickey, and J. Duffy. Influence of item content and stereotype situation on gender differences in mathematical problem solving. *Sex Roles*, 41(314):219–240, 1999.

[35] S. Trechter. A marked man: The contexts of gender and ethnicity. In J. Holmes and M. Meyerhoff, editors, *Handbook of Language and Gender*. Blackwell, 2003.

[36] G. M. Weinberg. *The psychology of computer programming*. Van Nostrand Reinhold, New York, NY, 1971.

[37] J.-M. Hoc, T. R. G. Green, R. Samurcay, and D. J. Gilmore, editors. *Psychology of Programming*. Academic Press, London, UK, 1990.

[38] R. Yates. Expert explanations of software. In *Psychology of Programming Work in Progress*, April 2011.

[39] M. Burnett, S. D. Fleming, S. Iqbal, G. Venolia, V. Rajaram, U. Farooq, V. Grigoreanu, and M. Czerwinski. Gender differences and programming environments: across programming populations. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 28:1–28:10, New York, NY, USA, 2010. ACM.

[40] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 3rd edition, 2006.

[41] Dimitri van Heesch. Doxygen manual, 2012. http://www.stack.nl/~dimitri/doxygen/.

[42] Ian Sommerville. *Software Engineering*. Addison-Wesley, 9th edition, 2010.

[43] Klaas G. van den Berg. Syntactic complexity metrics and the readability of programs in a functional computer language. In F.L. Engel, D.G. Bouwhuis, T. Bosser, and G. d'Ydewalle, editors, *Cognitive Modelling and Interactive Environments in Language Learning*, volume 87 of *NATO Advanced Science Institute Series*, pages 199–206. Springer, Berlin, Germany, August 1992.

[44] Raymond P.L. Buse and Westley Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering (TSE Special Issue on the ISSTA 2008 best papers)*, 36(4):546–558, 2010.

[45] Barry Boehm and Victor R. Basili. Software defect reduction top 10 list. *Computer*, 34(1):135–137, Jan. 2001.

[46] Steve McConnell. Software best practices (blog), 2012.

[47] Joel Splotsky. Joel on software (blog), May 2005. entry titled "Making Code Look Wrong", accessed March 2012, http://www.joelonsoftware.com.

[48] Jeff Atwood. Coding horror: programming and human factors, May 2006. entry titled "Code Smells", accessed March 2012, http://www.codinghorror.com/blog/.

[49] Roedy Green. How to write unmaintainable code, 1997. http://thc.org/root/phun/unmaintain.html.

[50] B. Kernighan and R. Pike. *The Practice of Programming*. Addison-Wesley Professional, 1999.

[51] Lori Pollock, K. Vijay-Shanker, David Shepherd, Emily Hill, Zachary P. Fry, and Kishen Maloor. Introducing natural language program analysis. In *Proceedings of the Workshop on Program Analysis for Software Tools and Engineering (PASTE '07)*, pages 15–16. ACM, 2007.

[52] Charles Simonyi. Meta-programming: a software production model. Technical report, PARC Technical Report CSL-76-7, Dec. 1976.