International Journal of English for Specific Purposes

English Use in Computer Programming: Analyzing CMC Discourse Among Computer Programmers

Joshua Lee[a], Fay Chen[b], and Wenli Tsou[c]*

[a & c] *Department of Foreign Language & Literature, National Cheng Kung University, Tainan, Taiwan, ROC*

[b] *Foreign Language Center, National Cheng Kung University, Tainan, Taiwan, ROC*

Biodata

Wenli Tsou is a Full Professor in the Department of Foreign Languages & Literature, and currently the Director of the Foreign Language Center at National Cheng Kung University, Taiwan. She received her PhD in Foreign and Second Language Education from the State University of New York at Buffalo, US. She is the project leader of the National Cheng Kung University ESP and CLIL programs. Her research interests include teacher training, ESP, English as a Lingua Franca, Content and Language Interacted Learning and English as a Medium of Instruction.

Email: wtsou@mail.ncku.edu.tw

Joshua Lee is a third-year Linguistics and TESOL PhD student at National Cheng Kung University. His main areas of interest are computer-mediated communication, language in media, and language in advertising.

Email: lee45276@gmail.com

Fay Chen is an Assistant Professor of the Foreign Language Center at National Cheng Kung University, Taiwan. She received her PhD from the same university. She is a member of the NCKU bilingual education project team, helping with CLIL teacher training and curriculum design. Her research

interests include teacher training, ESP, English as a Lingua Franca, Content and Language Interacted Learning (CLIL) and English as a Medium of Instruction (EMI).

Email: faychen77@gmail.com

Abstract

As computers and software become increasingly ubiquitous, the need for computer programmers rises. The nature of computer programming allows people from all over the world to communicate and collaborate on projects together regardless of native language. Much of this is done in English. Although non-native English speakers have access to English for Specific Purposes (ESP) materials to help them learn English for information technology, there are few sources developed specifically for teaching computer programmers the language they need to work with peers. In order to develop instructional materials that mirror English usage among programmers, one must first collect real-world discourse and analyze the communication that takes place. This process involves discovering the vocabulary, grammar, and other language patterns that are used in a specific environment. This paper analyzes real-world discourse among computer programmers using Github issues. Online discourse between computer programmers was found to fall into five main categories with most language concerned with giving additional information about an issue, giving suggestions about how to fix an issue, and requesting additional information about an issue. Additionally, relatively little technical vocabulary was used. We use these findings to develop suggestions and main topics for an English for Computer Programmers course. We provide Main topics such as Giving Opinions and Discussing Future Events with can-do statements and examples from the discourse. These findings can aid lesson planners developing materials for English for Computer Programming.

Keywords: ESP, CMC, English for computer programming, discourse analysis, Github

Introduction

Computer programming is becoming an increasingly popular skill in many countries. The United States Bureau of Labor Statistics alone states that the increase in software developer positions is “much faster than average” compared to other jobs (Occupational Outlook Handbook - Software Developers, 2018). Software developer is just one job that uses computer programming. Because English has become the dominant language in science, academia, and business, it makes sense to study how English is used in the context of computer programming (Swales, 2000). The focus of this paper is to examine how computer programmers communicate using computer mediated communication (CMC) and how the findings can be used to assist educators that are teaching English for Specific Purposes (ESP) to non-native English speakers.

The Importance of ESP

Traditional English language learning in the classroom is often concerned with General English. While this type of English learning may be beneficial for using English for day-to-day tasks, the language used in General English may not be adequate for performing tasks in work environments (San & Suan Choo, 2017; Lee, 2016). Trace, Hudson, and Brown (2015) state that “Language for specific purposes (LSP) courses are those in which the methodology, the content, the objects, the materials, the teaching, and the assessment practices all stem from specific, target language uses based on an identified set of specialized needs” (p. 2). Complaints from students and employers about graduates not being able to communicate in English have led to ESP programs which aim to better equip students with the specific English skills that they will need in their careers (Rackeviciene, Janulevičienė, & Mockiene, 2019; Al-Tamimi & Lin, 2010). Studies like these show researchers that knowing General English is not enough for many learners. Instead, many English as a Second Language (ESL) and English as a Foreign Language (EFL) learners require additional English training in the form of ESP courses.

Previous studies have looked at English for Specific Purposes in the context of computer programmers (Mykytenko, Rozhak, & Semeriak, 2019; Danielle, 2015); however, these studies have not looked at authentic English language used by programmers in a CMC environment. We believe that this type of discourse should be examined as written CMC is common among computer programmers.

English in Computer Programming

English has become the predominant language used when discussing science and technology. Academics and educators have used the terms Scientific English or English for Science and Technology (EST) to describe the language used in these fields (Tarantino, 1991). Like other ESP contexts, English use in science and technology is often much more context specific than EGP. Tarantino states that "common sense" words often used in EGP such as *far, heavy, cold* and *hot* have much more specific meanings in ESP settings. Therefore, it is important for a learner to recognize how these words are used in ESP settings compared to EGP settings.

To make matters even more difficult for ESL and EFL learners, much of computer programming relies on using English words to write code and markup. English usage in computer programming can be broken up into two types: computer-read English and human-read English.

Many computer-read languages' syntax have reserved English words that must be used for the computer to understand commands. Reserved words are different across programming languages, but common words are *if, while, for, switch, break, continue, next, using, class* and *initialize*. While there are some programming languages that use non-English syntax, *Latino* and *Ezhil* for example, most popular programming languages use English syntax. In December 2018, TIOBE, a software company that does market research concerning the software industry, reported the twenty most popular programming languages based on current usage data. All twenty of these languages' syntax use English words (TIOBE Index for December 2018, 2018). Additionally, there are organizations that decide which technologies are to be used in different computer programming environments. The World Wide Web Consortium (W3C) sets web standards for how HTML and CSS, two languages used in web development, are used to build websites (W3C Standards, 2018). Both of these languages rely heavily on English keywords. For example, HTML contains English keywords *head*, *body*, *aside*, and *main* while CSS contains keywords such as *background-color, text-decoration*, and *font-family*.

Human-read English is English that is meant for the programmer writing the code or other programmers that wish to collaborate with the original author. Computers generally do not understand human-read code. In computer programming, variables are containers for information and stored by naming them. For example, if a programmer was writing software for a banking application, he or she could use a variable like so:

```
currentBalance = 1000;
```

An additional way that human-read English can be used in computer programming is commenting code. Comments are notes that a programmer writes in his or her code to remember things or to help other programmers that are reading the code. One example is:

```
checkCurrentUserAccountBalance(); // this function checks the current
balance and returns it
```

This type of English, compared to computer-read English, requires a higher level of English proficiency as it requires both productive and receptive uses of English and requires the user to be able to express complex ideas that will have to be understood by another human.

English to Be Learned in ESP

Previous research has been conducted examining the language used in Information Technology (IT) roles (Balaei & Ahour, 2018; Synekop, 2018); however, research like this does not provide specific language that is used in real-world examples, something that should be used if the goal of a class is to teach authentic language. Current texts exist that focus on teaching English for Information Technology (Evans, Dooley & Wright, 2012; NCKU ESP Program, 2011) do contain language points that can be used to create lesson objectives in an ESP course; however, texts like these are much more wide-angled, focusing on English used in information technology as a whole instead of specific language used by computer programmers. Relativity few texts exist that specifically focus on teaching computer programmers.

Although knowledge of grammar and vocabulary are important when communicating in an ESP environment, other factors are equally important. In a study examining EFL customer service representatives in the Philippines, Forey and Lockwood (2007) note that "it appears that communication failure has less to do with the traditional notions of poor language skills, i.e. poor grammatical knowledge and poor pronunciation and more to do with poor interactional discourse skills and cultural appreciation" (p. 323). In order to promote communicative competency in addition to purely linguistic competency, discourse skills are taught in some English classes; however, sometimes, specifically for English used in workplace environments, the discourse that is taught does not parallel language that would actually be used in a real-world situation (Bremmer, 2010). That is, even when studying ESP discourse in the classroom, there is still a gap between what is taught in the classroom and what is used in the workplace (Bremmer, 2010). For example, Bremmer (2010) states that discourse found in ESP textbooks often involves idealized work environments and situations in which equity is equal among all participants, two situations which are not common in actual work environments. In order to close this gap between the two environments, we propose using discourse from real-world examples to create the target language for an English for Computer Programmers curriculum.

Research Questions

The research questions that this paper intends to answer are (1) What English do computer programmers use to communicate online and (2) what language skill training should an ESP course for computer programmers provide? We hope the findings of the study provide educators with authentic language that can be used in course design.

Method

Materials

In order to gather discourse to analyze, we needed to find a suitable dataset that included communication between programmers. A similar method has been used before to help develop ESP writing (Zhang & Hu, 2010). Additionally, we wanted to provide specific examples in our data. As such, we looked for a source of discourse that could be easily accessed and shared without worrying about using public information. We chose to gather data from Github. It should be noted that we only chose Github because accessing this data was convenient. Github (github.com) is a well-known website that allows programmers to store code and collaborate with other users. Over 31 million developers and over 2.1 million organizations worldwide use Github. Additionally, the largest number of contributors on Github come from outside America (The State of the Octoverse, 2019). While there has been previous research about communication between computer programmers, little research has been done to examine the kind of English used on Github. If the purpose of teaching ESP is to teach the learner how language is used in the target environment, examining Github communication is the best way to accomplish it.

The two main methods of communicating on Github are pull requests and issues. Pull requests are focused on the development of new code or features of a project while issues are typically focused on bugs that occur in the already written code. The focus of this paper will focus on the latter, communication concerning fixing bugs, for the following reasons. The discourse in pull requests is often short, containing just one comment from a single person. If pull requests were to be analyzed, there would be fewer chances to view how commenters communicate with each other in longer discourses. In addition, the discourse in pull requests is much more technical than discourse in issues. In order to understand the discourse in pull requests, a coder would have to be familiar with the project. Although, familiarity with a specific project is helpful when viewing discourse contained in issues, many of the problems in the issues category are much more general. The problems in issues may be technically related, but programmers without an in-depth knowledge of the specific project would still be able to understand the text. Finally, issues provide a narrower analysis than analyzing pull requests.

In order to get a variety of discourse, we chose one thread from ten different projects from the "trending of the month" projects during April 2019. All of the threads were closed threads which means no more comments would be added. Additionally, all of the projects that we chose had their descriptions written in English. Finally, all threads were about projects that used the Ruby programming language. Different programming languages will largely have the same content, but we are familiar with the Ruby programming language, so we chose this language.

Each comment in the selected threads belonged to one author; however, one author could write multiple comments. A total of 67 authors were gathered from the set of comments. Each user on Github has the option to list his or her location. Unique locations and the number of participants in those locations are as follows: Not listed: 19, USA: 17, Germany: 6, Canada: 3, France: 3, Scotland: 3, Australia: 2, Japan: 2, UK: 2, Indonesia: 1, England: 1, Finland: 1, Lithuania: 1, Norway: 1, Philippines: 1, Poland: 1, Switzerland: 1, Thailand: 1. It should be noted that these locations are not indicative of a user's native language.

Procedures

Shanthi, Wah, and Laijum (2015) state what when coding and categorizing discourse data, the researcher can approach the text in two different ways: (1) approaching the data without pre-created categorizes and allowing categorizes to emerge from the discourse or (2) approaching the data with a list of coding categories, allowing the researcher to place the discourse into each category as he or she sees fit. The first method will be used in this paper to allow the researcher to approach the data without any preconceived ideas of what kinds of moves the text contains.

We read over all of the comments in the project threads and broke them up into different discourse units. For example, a single comment or a single sentence can include requesting information as well as giving an opinion about a solution. We broke up comments like this into multiple discourse units where each unit constituted one action.

After breaking up the text into discourse units, we read through the data to discover various contexts, moves, and steps. After looking over about half of the data there became a point when no new discourse types emerged. At this point, we gave contexts, moves, and steps definitions. Two of us used these definitions to separately tag half of the data in order to get a more reliable perspective about which discourse units fell into which categories. We agreed 89% percent of the time out of 197 discourse units. After establishing a suitable inter-rater reliability, one researcher coded the remaining data on his own.

Results

The ten threads included in the discourse contained 398 discourse units. The discourse analyzed in the data fell into five different general contexts, each having their own moves and steps. Some moves were simple and included just one step, while other moves were more complex and included multiple steps. Although these moves and steps are presented in an order, many times users did not follow a particular order. The types of moves and steps that were found in the discourse and the number of times they appeared in the data can be seen in Table 1. As indicated in Table 1, the discourse found in the text fell into five main contexts: (1) seeking help, (2) a user responds to a question or request, (3) the problem-posted user responds to a question or suggestion, (4) users discuss future fixes, (5) and forum management. Each of these contexts is further discussed below.

Table 1: Moves and Steps

Discourse type	Number of occurrences	Example
Context 1: Seeking Help	-	-
Move 1: Provide background information	-	-
Step 1: Describe the problem	14	"I'm writing a new admin/scanner/ftp module."
Step 2: State previous steps taken that caused the problem	9	"This is true of any {CODE} parameters I've tried {CODE}
Step 3: List previous outcome	19	"Whenever I run exploit on my other ftp_traversal modules: {CODE}"
Step 4: Provide additional information	20	"This is with my git updated repo (as of today) as well as my kali apt-get pulled version."
Move 2: List steps taken to fix the problem	5	"Accordingly I installed Jekyll-paginate gem…"
Move 3: Ask for suggestions, help, opinions or more information	8	"Please could someone help me with this issue?"
Context 2: A User Responds to a Question or Request	-	-
Move 1: Ask for additional information	21	"Did you copy any code?"
Move 2: Discuss similar experience	-	-
Step 1: State previous experience	11	"I've run into the very same issue this morning."
Step 2: Provide background information	7	"When I received my verification email I got his error when I click the confirmation link."
Step 3: Can't replicate issue	2	"No wonder I couldn't duplicate."
Move 3: Respond to a question or give information about the problem	28	"{CODE} hasn't been complete fixed yet."

Move 4: Give suggestion or solution	-	-
Step 1: Give instructions	34	“In the meantime you should manually add the staging directory via the Alfred GUI.”
Step 2: Discuss suggestion or solution or give information about the suggestion or solution	15	“The syncfolder is only ever set if the user has set a sync folder, which may be never. If a syncfolder has never been set (or the user is a non Powerpack user), you can just assume {CODE} will be located in /Library/Application Support/Alfred 2/”
Move 5: Refer user to another thread or to other comments	13	“FWIW, I had a {LINK} suggesting Alfred find symlinked apps in their search scope which naturally includes /Applications by default.”
Context 3: Person with a Problem Responds to Question or Suggestion	-	-
Move 1: Show appreciation	12	“Thank you for stopping by.”
Move 2: Clarify or confirm understanding	8	“but should one plugin failure make whole server down?”
Move 3: Give opinion about suggestions or solution	9	“I don’t buy that Arel is a private API for Rails, given that Arel objects are documented as part of the public ActiveRecord API.”
Move 4: Report outcome	-	-
Step 1: List steps taken to fix the problem	18	“I added this code snippet on my_config.yml”
Step 2: Report positive outcome	10	“I just fix it.”
Step 3: Report negative outcome	21	“It doesn’t work.”
Step 4: Further discussion of solution	7	“It only suppresses specific messages.”

Move 5: Respond to a question	17	"i did, but it was from the bison_ftp_traversal module."
Move 6: Give additional information about the problem	15	"also, seems my wifi driver crashes whenever i wireshark or tcpdump…"
Context 4: Discuss Future Fix	-	-
Move 1: State intent to fix in the future	9	"I'll just add these two folders to Alfred's default search scope in 2.6."
Move 2: Propose a new fix or solution	16	"To improve the UX a little, I suggest that we…"
Move 3: State a new fix has been created	2	"I've now added these folders into Alfred's 2.6 default search scope."
Move 4: Give opinion about a fix or suggested idea for fix	40	"I'll reiterate, however, that just including their default directories in Alfred's results by default might be the best solution for the amount of work it requires."
Context 5: Forum Management		
Move 1: Thread status	4	"Anyone still encountering this should open a new issue."
Move 2: Close conversation	3	"Feel free to close and create another issue if you want."
Move 3: Forum rules and etiquette	1	"Please be nice."

Context 1: Seeking Help

The first context in the discourse is concerned with a user that has a problem. All threads start with this kind of discourse and other users that have the same problem may enter the thread after the conversation has started. Each move and step is discussed below.

Move 1: Provide background information

In this move, the user provides the forum with information about his or her problem.

Step 1: Describe the problem

Here, the user describes the task that he or she was doing while the problem occurred. Text found in this discourse type include "After untapping the emacs tap, but still having…" and "I'm writing a new admin/scanner/ftp module…"

Step 2: State previous steps taken that caused the problem

This step includes the user stating what specific steps he or she did to cause the problem. Sometimes these steps are listed as bullet points and as incomplete sentences, while other times the steps are written out in complete sentences. Text found in this discourse type include "I installed (software package)..."

Step 3: List previous outcome

This step involves the user stating what the problem or error is. Many times, this step is accompanied by a block of code or a screenshot. Text found in this discourse type include "Whenever I run exploit on my or other ftp_traversal modules: (screenshot)"

Step 4: Provide additional information

In this step, the user with the problem provides any additional background information about the problem including hardware or software configurations. Text found in this discourse type include "Here are my alfred preferences (block of code)".

Move 2: List teps taken to try and fix to problem

This move involves stating the steps taken to attempt to fix the problem. Sometimes, the steps are listed out in bullet points and not in complete sentences. Text found in this discourse type include "I installed jekyll-paginate gem…"

Move 3: Ask for suggestions, help, opinions, or more information.

This move involves the user with the problem explicitly asking for assistance. Text found in this discourse type include "Please could someone help me with this issue?"

Context 2: A User Responds to a Question or Request

The second context that was found in the data was when a user was responding to another user that had a problem. Each move and step is discussed below.

Move 1: Ask for additional information

Here, the person is requesting additional information about the problem in order to help. The question can ask about previous actions the other user took, or the question could be about what configuration the other user has. Text found in this discourse type include "Did you copy and code," and "what version are you using?"

Move 2: Discuss similar experience

The user that is trying to help previously had the same problem or an issue similar to the problem the original user currently has, but the user in this move no longer has the problem.

Step 1: State previous experience

The user states that they previously had the same or similar problem. Text found in this discourse type include "I was doing something similar."

Step 2: Provide background information

The user gives additional information about his or her previous experience. Text found in this discourse type include "When I received my verification email I got this error when I clicked the confirmation link."

Step 3: Can't replicate issue

This user does not currently have the problem, but he or she is trying to cause the same or similar problem for debugging purposes. Text found in this discourse include "Strange, I can't repo."

Move 3: Respond to a question or give information about the problem

Here, the user responds to a question or gives information about the problem. Text found in this discourse type include "... are not documented anymore. They were there by mistake."

Move 4: Give suggestion or solution

In this move, a user is helping another user that has a problem by giving that person specific suggestions or solutions that can alleviate the issue.

Step 1: Give instructions

The user gives specific instructions about how to fix the problem. Text found in this discourse type include “add to your gem file {CODE}” and “You can do this instead {CODE}.”

Step 2: Discuss suggestion or solution or give information about the suggestion or solution

Many times, after giving a solution to a problem, a user will give additional information about the solution. This information is usually about why the proposed solution works or an opinion about the solution. Text found in this discourse type include “I’ve found the ‘real’ bug, but that workaround is ‘good enough’ for now :)” and “{CODE} are not documented anymore. They were there by mistake {CODE} is also private API.”

Move 5: Refer user to another thread or to other comments

In this move, a user recommends that the user with the problem goes to another webpage on Github or outside of Github in order to help with the problem. Text found in this discourse type include “Did you follow instructions on {LINK}” and “This might help you {LINK}.”

Context 3: Person with a Problem Responds to Question or Suggestion

This context involves the user with the problem, however, at this point the problem has already been stated and suggestions have been given or questions have been asked. In this context, the person with the issue is responding to others. Each move and step is discussed below.

Move 1: Show appreciation

The user thanks another user for their suggestion or just for trying to help them. Text found in this discourse type include “Thanks for helping me” and “Cheers.”

Move 2: Clarify or confirm understanding

The user has received a suggestion or question and he or she is trying to clarify or confirm his or her understanding. Text found in this discourse include “Are you saying that I need to…” and “How do I do that?”

Move 3: Give opinion about the suggestion or problem

The user has received a suggestion or solution for his or her problem and he or she is giving their opinion about it. Usually, this occurs before a user has tried the solution. Text found in this discourse include “I don’t think that will fix my problem.”

Move 4: Report outcome

The user has been given a suggested solution and the user has tried it. Here, the user gives information about his or her attempt to fix the problem.

Step 1: List steps taken to fix the problem

The user lists the steps that he or she did to address the problem. Text found in this discourse include “I did…,” I tried,” and “Doing….”

Step 2: Report positive outcome

The user has tried a solution that has fixed the problem. Text found in this discourse include “Fixed!” and “It works now.”

Step 3: Report negative outcome

The user has tried a solution, but the solution did not fix his or her problem. Text found in this discourse include “Still getting the error” and “that didn’t fix my problem.”

Step 4: Further discussion of solution

The user has tried a solution and wants to further discuss the fix. This can occur if the solution fixes the issue or if it doesn’t. Text found in this discourse include “This should be fixed with the main code.”

Move 5: Respond to a question

The user is asked a question and he or she responds. Text found in this discourse include “I am using Windows 10.”

Move 6: Give additional information about the problem

The user gives additional information about the problem, but this is not a response to a question. Text found in this discourse include “By the way, I’ve tried doing this on Chrome and Firefox.”

Context 4: Discuss Future Fix

Some of the problems that appear in the discourse are due to the actual software package itself, not because of a user error. As such, a solution to the problem must be implemented by the project maintainers. Each move and step is discussed below.

Move 1: State intent to fix in the future

A project maintainer, or someone else that is able to modify the project's code, states that he or she will fix the issue in the future. Text found in this discourse include "I'll probably revert the fixed width this week."

Move 2: Propose a new fix or solution

Someone proposes a new fix to the solution. This is not giving advice to the specific user that has the current problem, instead this is a proposal to fix the core code that will fix all similar problems. Text found in this discourse include "Maybe there should be…" and "How about we add…"

Move 3: State a new fix has been created

A modification to the project's code has been made that should fix the user's problem. Text found in this discourse include "There's a new fix" and "This has been fixed with the new update."

Move 4: Give opinion about the fix or a suggested idea

A fix has been suggested or implemented and this user is giving his or her opinion about it. Text found in this discourse include "The fix works great" and "I don't think that will be a great idea."

Context 5: Forum Management

At some points in the threads, comments about the thread itself were given. Each move and step is discussed below.

Move 1: Tread status

A user asks or gives information about the status of the thread. The user can also make a suggestion about the status of the thread. Text found in this discourse include “What is the status of this.”

Move 2: Close conversation

A user closes the conversation. When this move appears, it is always the last comment of the thread, but this move doesn’t occur in every thread. Additionally, many times a bot automatically closed the thread after it was inactive for a period of time. Text found in this discourse include “Closing this as it is resolved as it will be.”

Move 3: Forum rules and etiquette

A user suggests how to communicate on the thread. Text found in this discourse include “Please be nice.”

In addition to the various contexts we discovered in this discourse, we found multiple sentence and grammatical patterns that frequently occurred. One pattern was using certain language to sound more colloquial. Many users used speech mannerisms such as “Ah yes” and “Ugh.” Additionally, many users also dropped the subject of a sentence such as “(I) sounder if there’s a way…,” “(I) Don’t think there is…,” “(It) looks like all of the {CODE} have the same issue,” and “ (This is a) very helpful analysis on the bug.” Another pattern that we commonly found was language used to give opinions and suggestions. Language used to give opinions and suggestions included “I suggest that we…,” “I think was should…,” and “Somebody could do…”

Discussion

In this section, we address each research question using data obtained from the discourse analysis.

English Used by Computer Programmers Online

Frequency used

As indicated in the result section, computer mediated discourse between computer programmers typically falls into five different contexts: seeking help, responding, problem-posted user responding, future fixes and forum management. Context 2, responding, was the most common with 32% of all discourse units followed by Context 3, problem-posted user responding, with 30%, Context 1, seeking help, with 19%, Context 4, future fixes, with 17%, and Context 5, forum management, with 2%.

The most common types of discourse found in the data involved asking and answering questions: Context 2 and Context 3. Over half of the discourse gathered included different members of each thread asking questions, answering questions, and discussing the answers to those questions. Context 3 was the second most used context and contained reporting what happened after trying a solution. Much of the language that was used in this context involved simple past tense. Additionally, language relating to attempts such as the word *tried* was frequently used.

The more frequent use of Context 2 & 3 isn't surprising since Github issues are about opening threads to request help from the community. Surprisingly, however, Context 4 about suggested fixes to the software packages themselves was also a common occurrence. Comments about future suggestions include asking opinions about hypothetical future fixes and discussing future plans, while comments about the current issue at hand focus more on past actions and current problems. Here, we see that verbs tenses are usually different depending on if the subject is the current issue or if the subject is the current software package.

The third most common context was Context 1. This context included describing a problem and giving background information about a user's hardware and software settings. Vocabulary such as *operating system*, *updating*, and *running* (as in "I am running Windows 10") were used in this context. Each thread included moves from Context 1 and these moves were almost always the first text in each thread. This is most likely due to the fact that when creating a new issue, users are often prompted how to format their first message.

Context 4 was the fourth most used context. This discourse involved discussing future fixes to the software package being used, not discussing a current fix that the user with the problem can use. Some knowledge of computer programming or familiarity with the project may be needed to tell the different between Context 4 from other contexts discussing the current problem.

The "Forum management" context, while significant enough to warrant its own categorical context, was used the least. One reason may be from the use of bots. Some projects on Github include bots that automatically close threads or perform other actions related to the status of the thread. Additionally, instead of a participant performing a speech act to close a thread, the participant could manually close the thread through certain options without requiring any additional text. That is, forum management was often done by administrative actions, not through actual discourse.

Word Choices

Examining the discourse closely, we can observe the specific language and word choices used. For example, we found ellipsis to be used frequently in the data. The two main types of ellipsis found in the data were: deletion of self-referential pronouns and deleting of determiners. Some examples of deleting pronouns referring to the author of the comment include “Been there, you just need to add…” and “Forgot to mention…” Examples of deleting determiners include “Alternatively, add to your Gemfile…” and “I have strange issue…” Because it is impossible to determine what the participants’ native languages are, it is not possible to tell if these are errors or stylistic choices from the authors. That said, some of the examples of ellipsis read as if they were used to sound more friendly and casual while others seemed like overt errors.

When explaining a problem, users used causal language to explain things such as the word *because*. When giving solutions users used language such as conditional sentences using “If…then…” phrases. Casual language was often seen in all contexts except for Context 5. Additionally, users used many modal verbs such as *should, can*, and *might* to talk about the possibility of a solution working.

An additional observation we found while looking at the data is the usage of words such as “ah,” “hmm,” and “uh.” It seems like these, with the previously mentioned ellipses, are used to make comments in the threads mimic speech typically used in face-to-face interactions.

Although there were almost 400 discourse units found in the data, programming-specific vocabulary was not as common as expected. Threads for each project included specific vocabulary related to each project; however, there was common vocabulary among the threads including the following: *operating system, updating, install, module, query, paginate, log*. With the exception of these words and project specific vocabulary, the range of programming-specific vocabulary was limited.

Languages Skills an ESP Course for Computer Programmers Should Provide

The above findings can help ESP teachers design at least the following three language skills to help their students: can-do statements, learning for low-frequency words, and grammatical structures. Can-do statements are often used in English language teaching to assess a learner’s abilities in regard to the target language (Denies & Janssen, 2016). Often, English language textbooks will include can-do statements and objective for each section or unit of the course.

These can-do statements are used in order to guide teachers with the content and activities they use in the classroom. The "can-do statements" we developed based on the findings of the study can be found in Table 2. This table contains main topics, expected learning outcomes, and real discourse for teachers to use in lesson planning. For example, a section about giving opinions would have can-do statements such as "the student will learn to use and understand sentences with I think/believe/suggest structures." An ESP teacher can use information in this table to develop role-plays and other suitable activities to teach the target language. The first row in Table 2 is titled "Giving opinions." This topic can be the theme of a role-play, unit, or single class depending on the students' needs. Specific goals and target language are also provided in order to guide the ESP teacher. Finally, authentic language from the discourse is also provided to give an example of how the target language is used between computer programmers. The multiple rows in the table can be used together to plan a complete English for Computer Programmers course.

Table 2: Can-Do Statements

Items	**Explanation/Expected Learning Outcomes**	**Examples from Data**
Give opinion about fix or suggested idea for fix		
Giving opinions	Students learn to use the sentence structure that begins with I think/believe/suggest…	"I feel that {SUGGESTION}. Is gonna have to wait a few years until it is worth the effort." "That would be the easy solution
Discuss future events	Students learn to use the sentence structure that begins with the recommended action. Students learn to use the future tense.	"Yes, adding to the error message that non-paid users can manually add {CODE} to the searched folders will help."
Use multiple verb tenses together sentence	Students learn to describe a past and a future action in one sentence.	"I've now added these folders into {SOFTWARE} default search scope. {SOFTWARE} should be out in a few weeks or so…"
Use conditionals	Students learn that the conditional sentence is also used when an opinion or recommendation is given.	"If we decided we need this, here's what I'd do: {CODE}"
Give instructions		
Giving instructions in a sequence	Students learn to give instructions in a specific order.	"If you're using {SOFTWARE}, shell into the container

		first…then get into postgres…Then try to select your user…”
Using sequence words	Students learn to use words to describe the order in which something happens.	“After untapping the emacs tap, but still having {CODE} installed. {CODE} fails with {CODE}.”
Giving commands	Students learn to give commands.	“Add these to the scope.” “Then try again please.”
Respond to a question or give information about the problem		
Thanking someone for helping	Students learn to thank another user.	“Thanks {USER} for researching and documenting this.” “Cool, thanks for the info!”
Replying to questions	Students learn to reply to previously asked questions.	“I did, but it was from the {CODE} module.” “That is correct.”
Using conditionals	Students learn to use conditional language.	“Ah yes, the preference doesn’t exist until I choose a sync folder.”
Ask for additional information		
Asking questions	Students learn to ask questions.	“Is there a way we can make linking work with the free version of {SOFTWARE}?”
Reporting outcomes		
Reporting previous steps taken	Students learn to use signal language to report previous steps taken.	“After fixing the typo in your SQL statement it looks like…”
Reporting outcomes	Students learn to report positive and negative outcomes of trying a proposed solution.	“Now running {CODE} works!” “And attempting to uninstall {CODE} itself.” “Strange, I can’t repro.” “I’m no longer getting this issue, thanks!”
Discussing previous attempts	Students learn to use language to describe past or habitual actions.	“Seems my wifi driver crashes whenever i wireshark or tcpdump.”
Using past tense	Students learn to use simple past tense to describe completed actions.	“I’ve updated this issue…” “I didn’t read the above discussion properly.”
Provide additional information		

Giving detailed information about a problem so someone can help	Students learn to use and understand complex sentences with embedded clauses.	“Running Rails in a separate window causes foreman to get killed, so I can’t test this live although taking those steps in Rails seems to be successful.
Thanking a user for helping	Students learn to use language to express gratitude.	“Thank you for stopping by, {USER}.”

Secondly, an ESP design can include reading strategies for obtaining the meaning of low-frequency words (Masrai, 2019). The discourse in the text included programming-specific vocabulary, but besides project-specific words, computer programming vocabulary was somewhat limited. Each project contained vocabulary that was specific to the programming language or software package that was being discussed. Therefore, it would be impossible for an ESP teacher to teach vocabulary for every programming language and every software package as the number of new languages and software packages increases each day. Instead, an ESP teacher can teach students reading strategies, such as context clues, to help students when they come across new words (Khabiri & Pakzad, 2012).

Finally, ESP teachers can help programming students by providing common grammar points that are used in CMC discourse. The discourse often included language that mimics spoken language using words like *ah* and *ugh*. Grammatical structures using *I think* and *I suggest that* were common in situations that involved talking about suggestions or opinions. Additionally, the omission of subjects such as the personal pronoun *I* and empty subjects like *it* and *there was* common. Students should be familiar with these grammatical structures as they occurred frequently in the dataset. It may be beneficial for ESP teachers to make their students aware of the differences between English used inside the professional circle of the programmers and general English outside of the profession.

This research does contain a few limitations. First, the programming language used in the discourse, Ruby, is most commonly used for web development. Although much of the language in web development will transfer over to other forms of programming, some fields such as game development and mobile development will have domain specific vocabulary and discourse. These subdomains also include language related to computer hardware as well as physics. That is, while this research is narrowly focused in terms of ESP, future research can be even narrower, focusing on subfields of computer programming. Second, all of the discourse was hand-coded by the researchers. This led to inherent time limitations. Some of the findings, such as lexical items that were used in the discourse, may be expanded on by writing scripts to pull out all of the comments' text, checking which lexical items are used, and comparing these items to a commonly used word lists to determine which vocabulary items would be beneficial to cover in an English for Computer Programmers course.

Conclusion

In this study, we examined English use in computer programing. Current educational materials for ESL and EFL learners regarding English use in programming is lacking, therefore current English usage among computer programmers was analyzed in order to examine communication patterns and techniques. After analyzing how computer programmers communicate with each other, we used the information to create suggestions for planning an English for Computer Programmers course.

We found five main categories in the CMC discourse: seeking help, responding to questions or requests, a user with a problem responds to a question or suggestion, discussing future fixes, and forum management. The most common context was responding to questions or requests. Each of these contexts includes multiple moves and steps such as describing processes and outcomes, providing background information on a specific problem, and proposing new fixes. Although computer programming is a technical field, only a small amount of the discourse included domain-specific vocabulary. Language usage included using words such as *ah* and *ugh* in order to closely mimic spoken language occurred frequently. Additionally, the dropping of the subject such as expletive pronouns *it* and *there* as well as the subject *I* when the subject is assumed to be understood was common.

These findings can help develop topics and can-do statements that will aid lesson planners when developing ESP material for English for Computer Programming courses. These suggestions, along with the findings of how language is used in real-world examples, fill a gap in current ESP research and can provide fruitful sources for ESP designs.

Acknowledgements

The authors would like to thank the participants of the study, the financial support from the Ministry of Science and Technology of Taiwan.

References

Al-Tamimi, A., & Lin, S. (2010). Investigating the English Language Needs of Petroleum Engineering Students at Hadhramout University of Science and Technology. *The Asian ESP Journal*, *6*(1), 6-34.

Balaei, P., & Ahour, T. (2018). Information Technology Students' Language Needs for their ESP Course. *International Journal of Applied Linguistics and English Literature, 7*(2), 197. doi:10.7575/aiac.ijalel.v.7n.2p.197

Bremner, S. (2010). Collaborative writing: Bridging the gap between the textbook and the workplace. *English for Specific Purposes, 29*(2), 121-132. doi:10.1016/j.esp.2009.11.001

Danielle, Joulia (2015). Facteurs d'échec et de réussite en anglais de spécialité: le cas de l'anglais pour l'informatiqueContributing factors to failure and success in English for Specific Purposes: The case of English for IT, Recherche et pratiques pédagogiques en langues de spécialité - Cahiers de l APLIUT, 10.4000/apliut.5239, Vol. XXXIV N° 2

Denies, K., & Janssen, R. (2016). Country and Gender Differences in the Functioning of CEFR-Based Can-Do Statements as a Tool for Self-Assessing English Proficiency. *Language Assessment Quarterly,* 13(3), 251-276. doi:10.1080/15434303.2016.1212055

Ellis, R. (1996). SLA and language pedagogy. *Studies in Second Language Acquisition*, 19, 69-92.

Evans, V., Dooley, J., & Wright, S. (2012). *Career Paths: Information technology*. Newbury: Express.

Forey, G., & Lockwood, J. (2007). "I'd love to put someone in jail for this": An initial investigation of English in the business processing outsourcing (BPO) industry. *English for Specific Purposes*, 26(3), 308-326. doi: 10.1016/j.esp.2006.09.005

Khabiri, M., & Pakzad, M. (2012). The Effect of Teaching Critical Reading Strategies on EFL Learners' Vocabulary Retention. *The Journal of Teaching Language Skills (JTLS), 4*(1), 74-106. Retrieved August 17, 2020, from http://jtls.shirazu.ac.ir/

Lee, C. L. (2016). Principles and Practices of ESP Course Design - A Case Study of a University of Science and Technology. *International Journal of Learning, Teaching and Educational Research*, 15(2), 94-105

Masrai, A. (2019). Vocabulary and Reading Comprehension Revisited: Evidence for High-, Mid-, and Low-Frequency Vocabulary Knowledge. *SAGE Open*, 9(2), 215824401984518. doi:10.1177/2158244019845182

Mykytenko, N., Rozhak, N., & Semeriak, I. (2019). Teaching Communication Strategies to The Computer Programming Students. *Advanced Education*, *6*(12), 49-54. doi:10.20535/2410-8286.167148

NCKU ESP Program. (2011). *ESP: English for information technology*. Taipei, Taiwan: Bookman Books.

Rackeviciene, S., Janulevičienė, V., & Mockiene, L. (2019). English for Specific Purposes and The Second Foreign Language: Reaching Beyond Language Training in *Ba Philology Study Programme. Journal of Teaching English for Specific and Academic Purposes*, 135. doi:10.22190/jtesap1902135r

San, N. Y., & Suan Choo, J. C. (2017). Needs Analysis of English for Technicians: A Case Study. *Langlit*.

Shanthi, A., Wah, L. K., & Laijum, D. (2015). Discourse Analysis as a Qualitative Approach to Study Information Sharing Practice in Malaysian Board Forums. *International Journal on E-Learning Practices (IJEL*P), 2. Retrieved 2015.

Swales, J. M. (2000). Languages for specific purposes. *Annual Review of Applied Linguistics*, 20, 59–76.

Synekop, O. (2018). Cognitive Aspect of Learning Style in Differentiated ESP Instruction for The Future It Specialists. *Advanced Education, 5*(10), 40-47. doi:10.20535/2410-8286.151271

Tarantino, M. (1991). English for science and technology: A quest for legitimacy. *English for Specific Purposes*, 10(1), 47-60. doi:10.1016/0889-4906(91)90015-o

The State of the Octoverse (Rep.). (2019). Github. doi:https://octoverse.github.com/

TIOBE Index for December 2018. (n.d.). Retrieved December 10, 2018, from https://www.tiobe.com/tiobe-index/

Trace, J., Hudson, T., & Brown, J. D. (2015). Developing Courses in Language for Specific Purposes. *Hawaii: National Foreign Language Resource Center*.

W3C Standards. (n.d.). Retrieved December 10, 2018, from https://www.w3.org/standards/

Zhang, Y., & Hu, J. (2010). A Genre-based Study of Medical Research Article Introductions: A Contrastive Analysis Between Chinese and English. *The Asian ESP Journal, 6*(1), 72-96.