Abstract

Many students face difficulty when writing documents due to various reasons such as language barriers, content misunderstanding, or lack of formal writing education. They are often too shy or busy to visit a writing center or speak with a professor during office hours. Technology also falls short in this arena. Asynchronous collaboration systems require students to self-report when they are struggling and many students tend to under-report difficulty. Synchronous collaboration systems eliminate the need for self-reporting, but require teachers to constantly monitor their students. By combining synchronous and asynchronous collaboration paradigms, this project is able to create a mixed-focus collaborative writing system in which students and teachers engage in collaboration only when triggered by an automatically or manually generated event that indicates the student is facing difficulty. This mixed-focus system was created by combining two existing architectures: 1) the EclipseHelper difficulty architecture for inferring programming difficulty, and 2) the Google Docs collaborative writing environment. The new, combined architecture allows teachers to intervene and offer remote assistance to their students when they are automatically notified that a student is facing difficulty. A user study was conducted to evaluate this new architecture. Students used the system to complete a two-page paper given in a class they were taking, and data were recorded during the writing and help-giving process. Using both qualitative and quantitative analysis, the data were evaluated. Overall, students found the help-giving model easy to use and appreciated the feedback they received. However, difficulty was predicted infrequently, likely as a result of inherent differences between writing and programming. Future work will involve further analysis of the data in order to improve the difficulty prediction algorithm.

Introduction

Many students face difficulty when writing documents. This is an issue faced by students of all ages, from middle-school students who are just beginning to learn about how to write an essay to graduate students writing lengthy research papers. Resources that help facilitate writing feedback often lack general-purpose remote real-time assistance capabilities, and this thesis aims to fill that gap. By combining existing collaboration paradigms and difficulty assistance architectures, this project creates a new architecture that allows teachers to intervene and offer remote assistance to their students when they are automatically notified that a student is facing difficulty. This paper discusses the development of this new architecture, a field-study created to evaluate the architecture, and an analysis of the results of the field-study.

Related Work

Benefits of Collaboration While Writing

Writing is crucial to education and it has been shown that writing improves understanding and critical thinking skills [1]. Giving and receiving feedback is important not only in helping students improve their writing skills, but also in helping them to develop skills related to personal judgment and responsibility [2]. Thus, it is important that students learn how to write well, and that they are able to receive instructor and peer feedback during the writing process.

However, while peer feedback is certainly helpful, students often feel more secure when they receive feedback from a teacher [2]. Gielen et al. noted that teacher feedback is more complex than peer feedback and, while their study revealed that students who receive teacher rather than peer feedback do not necessarily perform better, students do interpret comments from a teacher as being more reliable [5, 6].

A system that allows students to receive regular feedback from both teachers and peers would then be ideal. It has been shown that these feedback systems lead to learning benefits. For example, Hanjani found that EFL (English as a Foreign Language) students benefit from collaborative revision [7]. Collaboration as a writing tool can also extend beyond just the revision process to collaborative content creation. Pair writing has been shown to improve accuracy [8] and future writing [9]. Collaborative learning systems that allow a tutor to provide real-time feedback to a student have also led to an increase in knowledge retention [10].

Existing Solutions and the Need for a Mixed-Focus Architecture.

A number of resources already exist for students who need help with their writing. There are several automatic natural language processing tools that focus on helping students correct their grammar and spelling errors [10, 11, 12]. Some of these tools are very sophisticated, and can be particularly helpful to ESL (English as a Second Language) students who might disproportionately struggle with syntactical errors. Several other tools can auto-grade assignments for content errors, although their capacity can be limited [13, 14].

When these tools fail, students can take initiative and seek help from an instructor. However, students are often too shy or too busy to do this. In high school or middle school, parents and teachers often have to reach out to students who are falling behind. Students may be overlooked or not receive the academic attention they need, especially if class sizes are large. At the college level, struggling students are expected to visit the writing center or speak with a professor during office hours. This extra step can be intimidating and time-consuming, especially for new students who are unfamiliar with campus resources, shy students who are nervous to approach a professor to ask for help, or students who are busy with other work and do not have time in their schedule for an in-person visit. Another problem with many of these resources is

that teachers and professors often refer students to help only after they do poorly on an assignment. It would be more beneficial for students to have access to help before receiving a bad grade so that they do not run the risk of falling behind.

For students who cannot seek in-person help, there exist asynchronous collaboration systems [15, 16], which provide mechanisms for students to receive feedback from a remotely located instructor. However, these systems do not allow for immediate feedback. In addition, requiring students to initialize asynchronous collaboration and self-report difficulty can potentially lead to both over-reporting and under-reporting of difficulties. Johannesen and Eide found that active students benefit more from computer supported learning environments than passive students, since passive students will tend to under-report difficulties [17]. Because passive students would also intuitively be less likely to attend office hours or go to a writing center for help, it is even more important for learning technology to target their needs. Several studies have shown that similar issues arise in cooperative programming environments; developers are less likely to ask each other for help when they are working in different locations [18], new programmers are often late to ask for help [19], and programmers of all levels reach out to teammates only as a last resort after exhausting other forms of help [20].

To enable real-time assistance and minimize the self-reporting issue, there exist several writing environments that allow for synchronous collaboration [21, 22]. In these environments, multiple peers or instructors can work together on one document to engage in collaborative writing and revision. However, for a teacher to be able to help a student in a moment of need, the teacher has to be watching the collaborative document in order to see that the student is struggling. This is time-consuming and not scalable even for small classroom environments.

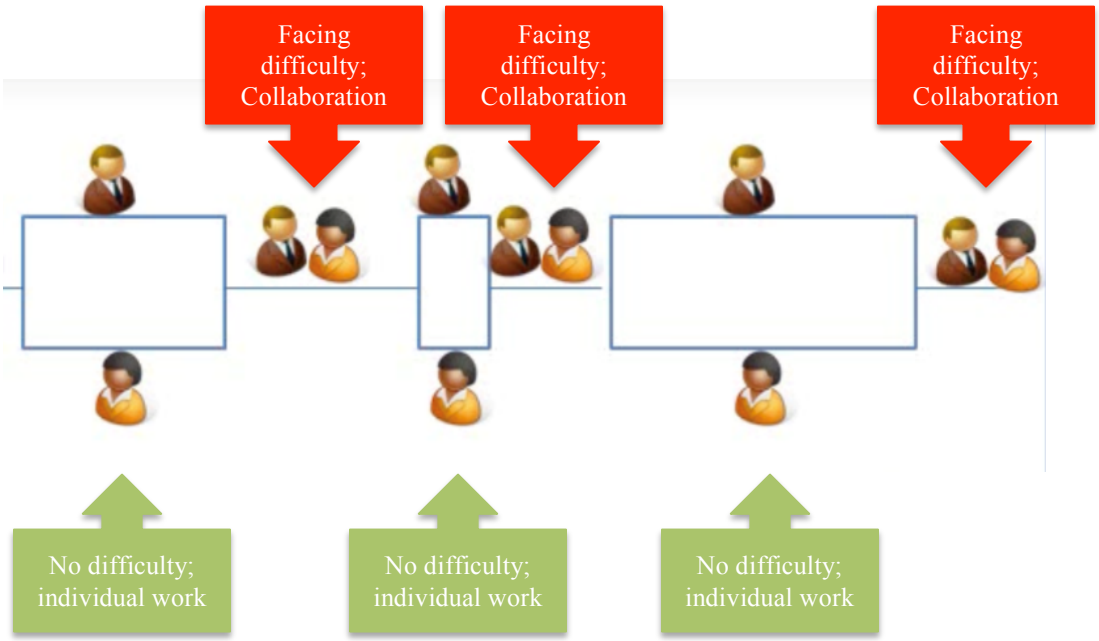

Figure 1: A Mixed Focus System

In order to remedy this problem and facilitate a learning environment that benefits passive students who are not co-located with the professor without placing unrealistic expectations on instructors, the author has focused on combining the synchronous and asynchronous learning environments into a mixed-focus learning environment. Dewan et al. discuss mixed-focus collaboration, defining it as an interaction mechanism that allows for both individual and group work to occur [23]. In a mixed-focus environment [Figure 1], students and teachers are able to work individually, only coming together to collaborate when a certain event occurs. Any event monitored by the collaborators can trigger this collaboration [23]. In a mixed-focus programming environment, this event is either an explicit request for help or a difficulty inference generated automatically [24]. In a mixed-focus writing environment, the event could also be triggered either manually or automatically. The automatic events prevent the problematic need for self-reporting that is present in asynchronous environments, and the system as a whole provides students with real-time assistance. In addition, it allows the teacher to manage students

in a more scalable manner than in a synchronous environment.

Architecture: an Interoperable Solution

Based on the earlier discussion, it is clear that a successful teacher-student writing environment needs several features. In keeping with the mixed-focus nature of the system, it needs to have a mechanism for inferring difficulty and then indicating to the teacher that the student is facing an obstacle. The student should also be able to manually report that they are facing difficulty. After the teacher receives notice of this event, the writing environment needs to facilitate conversation and collaboration between the teacher and student so that the difficulty can be resolved. In order to reuse existing tools for communication, collaboration, notification, and difficulty inference, the author combined existing tools and associated architectures to create a new interoperable tool and architecture. This architecture isolates what is different about this project, while identifying and reusing what is shared with other works.

Difficulty Inference Algorithm: EclipseHelper

The first existing tool that the author sought to incorporate was EclipseHelper, which has been implemented to assist students with programming assignments. It solves the self-reporting issue present in asynchronous assistance platforms by predicting when student programmers are facing difficulty [24].

The EclipseHelper algorithm looks at ratios of aggregate commands entered by the student in order to predict whether the student is facing difficulty. Commands such as typing a character or navigating between files within Eclipse are mapped to broader categories such as Edit or Navigation. After a certain number of commands are collected, a prediction is made using the ratios of each command category to the total number of commands collected in that segment. Segments are defined by number of commands collected, not time passed. Using these

ratios, a prediction is made [Figure 2]. Several parameters can be adjusted in generating these predictions. The segment length (the number of commands collected before a prediction is made) and startup lag (the number of commands the algorithm ignores before making the first prediction) can both be modified.

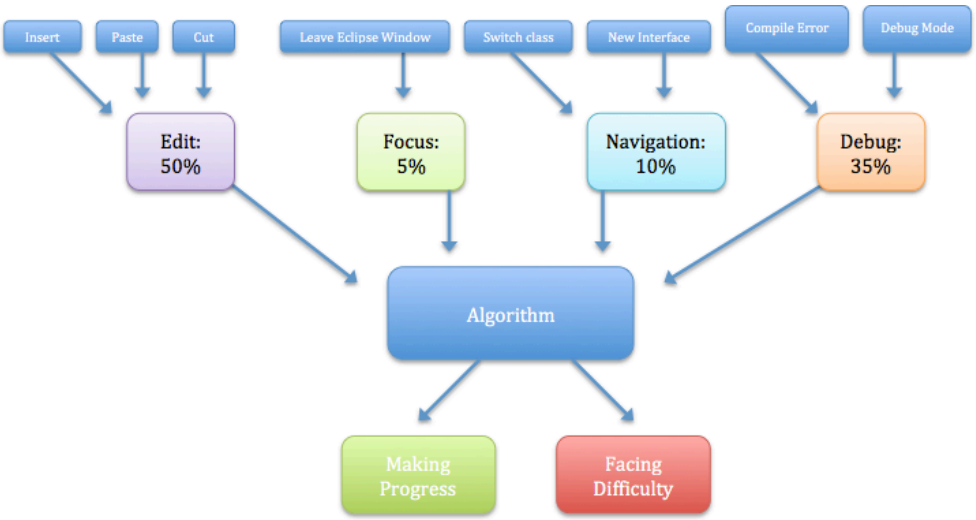

Figure 2: EclipseHelper Architecture

The EclipseHelper algorithm is still being refined, but it has been tested and improved over the course of several user and field studies [24, 25]. Students who have used the plugin have given it positive reviews, saying that it helped them to solve time-consuming problems and that they appreciated using the tool [25]. For these reasons, this project builds upon the EclipseHelper difficulty inference algorithm in an attempt to create a similar environment for document writers.

Collaboration and Communication: Google Docs

As mentioned earlier, a writing environment also needs to facilitate collaboration and communication between the teacher and student. One successful writing environment that already does this is Google Docs. Several key features made Google Docs more suitable for this study than other commonly used writing environments. First, Google Docs is already a widely used platform amongst students of all ages. Using this framework instead of creating a custom

editor minimizes the learning curve for users by allowing them to continue to use a platform with which they are already comfortable. Google Docs also runs on the web, allowing students to work on their document from any computer.

Second, previous research [26] indicates that software agents are more effective when they are presented as "peer companions" because receiving peer rather than instructor feedback reduces anxiety and fear of failure. This is supported by previous research on the benefits of peer editing [5]. Since Google Docs has a proven history of aiding peer-to-peer collaboration, the author hypothesized that students who are accustomed to collaborating with peers in the Google Docs environment will be more open to working with a teacher or a helpful computer agent through the same interface. This new interface will combine the anxiety-reducing experience of receiving peer feedback with the benefits of instructor feedback mentioned in the Introduction. To ensure that the teacher and software agent are presented as "peer companions" [26], the plugin UI that I created seamlessly integrates with the Google Docs interface [Figure 3].

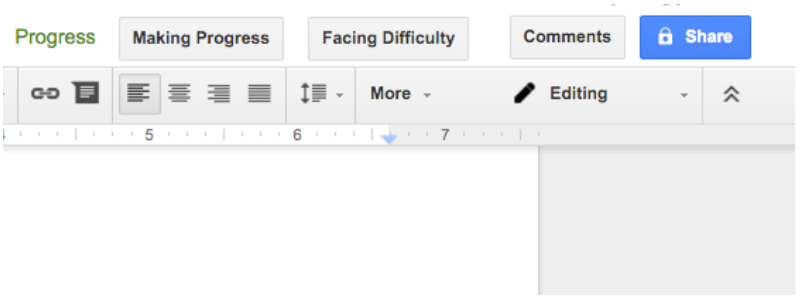

Figure 3: Google Docs User Interface

Finally, Google Docs allows external plugins to extend its functionality. Google Drive has an easy-to-use API, and Google Docs already tracks every revision made to a document for collaborative purposes. The author built upon an open-source project called Draftback [27], which uses a Google Chrome extension to examine the history of a Google Doc in order to replay the entire history of revisions to the document. This revision-tracking ability provides the basis for integration with the EclipseHelper algorithm, which will be discussed in more detail

later. For all of these reasons, Google Docs proved to be a suitable writing environment to assist with collaboration and communication between teacher and student.

The Interoperable Architecture

To combine the difficulty inference architecture of EclipseHelper with the collaboration and communication architecture of Google Docs, the author created a new architecture to facilitate a difficulty-triggered collaborative writing environment. In this new architecture, user commands are collected from Google Docs, mapped to EclipseHelper command categories, and passed into the EclipseHelper difficulty inference algorithm [Figure 4]. Once a prediction is generated, it is sent back to Google Docs, where the student can choose to correct it or continue with their work. The teacher is notified via email if the student is facing difficulty [Figure 5].

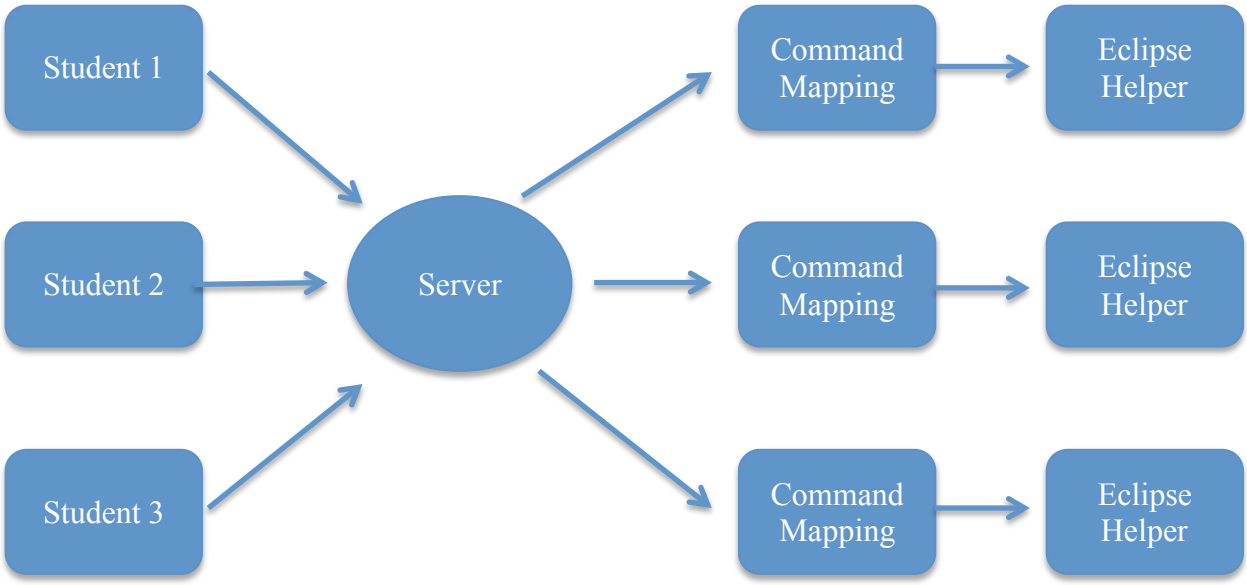

Figure 4: Interoperable Architecture Inner Workings

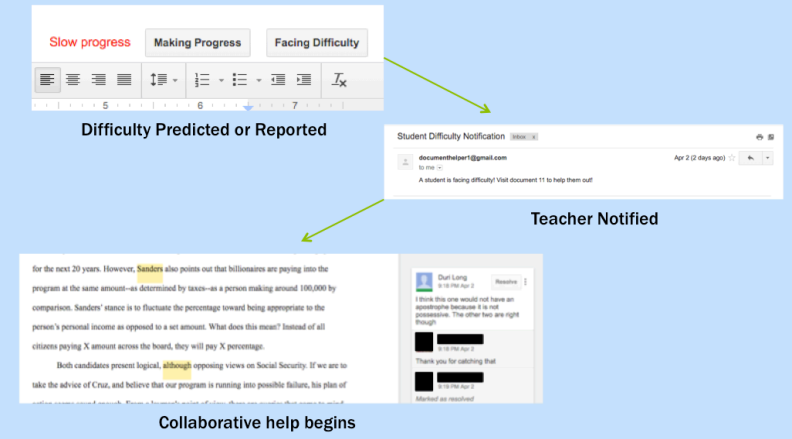

Figure 5: Interoperable Architecture User Experience

Implementation

Creating the new interoperable architecture required several additions to the EclipseHelper architecture. EclipseHelper was originally designed as an algorithm for individual use, so the author had to create a server to allow data from multiple students to be processed at once. In addition, the author had to modify prediction settings to make the algorithm work for student writers. When initially testing the tool, it rarely predicted difficulty. This was because of the segment length and startup lag. The typical programming assignment takes far longer and is a more command-rich experience than writing a two-page paper, so waiting 50 commands to make a prediction (startup lag) and then making predictions only every 25 commands (segment length) was not working. As a result, the author shortened the startup lag to 10 and the segment length to 5. Finally, writing and programming involve different commands and therefore customized command mappings are necessary. These command mappings are used to map document-specific commands collected within Google Docs to broader EclipseHelper command categories (Figure 6).

Creating the modifications to these command mappings was a two-step process. First, document commands had to be collected from the Google Docs writing environment. The creator

of the Chrome extension Draftback noted that every time a user types something in a Google Doc, a web request is sent from Google that indicates exactly what was edited and what type of edit was made (insert, delete, style, etc.) [27]. These requests are then stored in a log file that allows Google to keep track of revision history and collaboration within a document. Draftback uses this log file to replay the revision history of the document; the tool created for this thesis does not use this log file and instead uses a Chrome extension to listen to the web requests as they are sent in order to allow for live processing of commands. The Chrome extension created for this thesis also extends beyond Draftback and listens to browser events in addition to Google Docs commands. Commands are registered when users leave the focus of the document they are working on, either by navigating to a new URL, opening a new tab, or leaving the browser. Commands are also recorded when users scroll through their document, move their caret, or use the spellcheck tool.

The second part of the creating a new command mapping involved mapping these document commands to EclipseHelper command categories. The categories that EclipseHelper can currently map to are Edit, Debug, Insert, Delete, Focus and Navigation. The Edit category encompasses both the Insert and Delete subcategories; depending on the mapping that the programmer chooses, either the Edit super-category or the Insert and Delete subcategories can be used to make predictions. The Focus category encompasses commands that relate to leaving and entering Eclipse, and the Navigation category encompasses commands that related to navigating between classes in Eclipse. Many of the command mappings used when editing a document are clearly analogous to those used when writing a program, but some of the less obvious mappings made in this implementation will be discussed here. These mappings are also visualized in Figure 6.

Mapping to the Debug category proved to be the most interesting problem, as document writing does not have commands that are directly analogous to using the debugger in Eclipse. Currently, both compile errors and use of the debug tool map to the Debug category in EclipseHelper. The author concluded that spelling and grammatical errors made while writing a document are analogous to compile errors in a program. Developing a counterpart for use of the debug tool was somewhat more complicated. It is somewhat intuitive that the process of debugging a program is analogous to revising a paper, but what does that revision process entail?

To gain a better understanding of this process, the author collected survey responses about college students' writing processes. Surveys were collected from seven different college students with a variety of different majors[1]. When asked how they noticed and/or resolved content and phrasing errors in their papers, many students stated that they would reread their paper several times looking for organizational, grammatical, and phrasing errors. Some students revised on a smaller scale, treating each paragraph or sentence as a mini-draft and revising it before reading over the paper as a whole. Many students said they would also read their essay out loud or ask a peer to read over it before submitting a final draft. Between a rough draft and a final paper, most students stated that they made a small number of edits to fix phrasing or grammatical errors, although some students made structural changes. One stated, "I often even print out my paper so that I can physically draw the rearrangements".

The findings from this small survey are supported by more extensive previous research on the revision process. Hayes and Flower present a model for the stages of document writing that consists of several stages: "planning (generating ideas and organizing them), sentence generation (producing a draft based on your plan), and revising (improving a draft)" [28]. These

[1] Dramatic Art, Business and Psychology, Economics and Computer Science, Political Science, Public Policy and International Studies, Anthropology, Religious Studies

processes do not occur separately; they are often interwoven throughout the process of writing a document [28]. The different approaches amongst students that were found in the author's study are also supported by previous research. Hayes and Flower note that expert writers tend to conceive of revision as a "whole-text" task [28]. They make large-scale structural changes and reformulate their paper in order to improve their argument. Novice writers, on the other hand, tend to make minor changes to words or phrasing, often revising on a smaller scale.

From these results, the author concluded that concrete actions like using collaboration tools to facilitate peer-editing, scrolling through a document to make minor edits in specific locations (in the case of a novice writer), and moving or copy/pasting large chunks of text (in the case of an expert writer) all occur during the revision state of the writing process. As a result, the author chose to map copy/paste commands that deal with more than 100 characters and scroll/move caret commands to the Debug category. The EclipseHelper algorithm does not currently support analysis of sequences of commands (i.e. scrolling to a new page and then making several small edits) or location-based commands (i.e. making a lot of edits in one paragraph), but this is an area for future exploration that is discussed in more detail in the analysis. The final command mapping that the author developed is visualized in Figure 6. The Navigation category is empty because scroll/move caret commands were mapped to the Debug category.

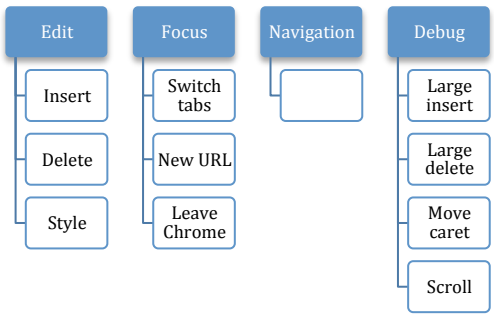

Figure 6: Command Mappings

Evaluation: A User Study

After conceptualizing and developing this system, questions still remained about how well the tool would fare in a real-world environment. Would students appreciate receiving unsolicited help, or would they find it unnerving? Would the EclipseHelper algorithm predict document difficulty as well as it predicted programming difficulty? Would students find the interface the author developed easy to use? Would they even want to install the tool? To find answers to these questions, the author conducted a field study.

This study enlisted students in two different Political Science classes (Introduction to Political Science and American National Government) at a two-year community college. The students were asked to complete an assignment for their course using the tool the author developed. The subject of the assignment was Social Security, and after being given some background, students were asked to write a response detailing their thoughts on the issue of Social Security, backing up their opinions with several outside sources. The author acted as the instructor, offering help to students when she received a notification that they were in difficulty.

Most of the students who participated in this field study are working full time jobs or caring for children in addition to pursuing their degree, and there were several non-native English speakers who participated. This was an ideal target audience for this tool, as it is designed to help students who struggle while writing but may not have the time to visit office hours for help.

Results

Qualitative Results

Student Results

After the students completed their assignments, they were given a survey asking about their reactions to the tool. Selected reactions are included in Table 1. When asked how they felt about receiving unsolicited help while writing, most students had positive reactions and found the help welcome. Students did not appear to have privacy concerns, and everyone appeared to feel comfortable being watched while writing. This is consistent with previous research that has found that students are willing to sacrifice privacy when they experience real-life benefits [29].

Students were also asked a more general question about what their help experience was like as a whole. Most students very much appreciated the help they received, and one student expressed that even though she did not receive help, she felt more confident in her writing process knowing that help was available if she needed it.

When asked about how easy the user interface was to use, all six of the students who participated successfully in the study said the user interface was either “Very easy” or “Somewhat Easy” to use. However, there are 12 other students who were originally enthusiastic about the tool and signed up for the study, but did not complete it. While some of these students likely just did not complete the assignment, at least three of the students struggled with the installation process and misunderstood parts of the instructions, causing their data to be invalid. Others probably faced similar issues, but did not make it as far in the process.

These are important problems that likely did not arise with the EclipseHelper field study because EclipseHelper was tested on a class of mostly technologically savvy Computer Science students. Introducing new technology to non-CS students proved to be more difficult and

although students were generally enthusiastic and excited to install the tool, the technological barrier served as a deterrent.

Finally, students were asked about suggestions for improvement to the tool. Students (accurately) felt that there were too few difficulty predictions made by the tool, an issue that will be discussed in more detail in the analysis. Several students also expressed a desire for natural language processing functionality to be added to the tool, so that they could receive better spelling, grammar, and phrasing feedback automatically. Potential future work could integrate GoogleDocs with a natural language processing architecture such as Grammarly [10], SpellCheckPlus [11], or CyWrite [12].

Q1: Unsolicited Help	**Q2: Help Experience**	**Q3: Usability**	**Q4: Suggestions for Improvement**
"The offer of assistance was surprising, but not unwelcome. It felt natural to be able to have the chat and comment thread available."	"Though I did not need help and didn't face any difficulties, I felt more confident in my writing process knowing that help was available should I need it. I feel that this would be very helpful to students, particularly in online classes, where they often are not able to establish open and more immediate lines of communication with the instructors."	"I have not used Google docs very often, and have relied on Word to type my documents for classes. The Google doc environment is much easier to navigate and the option of communication to receive assistance is invaluable."	Several students felt that the predictions were inaccurate. Students thought that long pauses, slow typing, and excessive spelling errors should indicate difficulty
"It was Welcome and Natural and nice to have someone right there to ask help from"	"I believe that this would be beneficial for both students and teachers. As a teacher, I like the fact that I could check in on my students and their progress, answering questions as needed. From a student standpoint, having help available when struggling on a paper is a wonderful option."	All six students reported that the tool was either "Very easy" or "Somewhat easy" to use	One student enjoyed receiving feedback on a paper after completion, rather than while they were writing.
			"If the tool highlighted grammar and spelling errors that would be helpful. And also when it reads that you are facing difficulty have it suggest solutions to the problem."
			"Maybe some kind of engine that I would put my sentence in and it would arrange it the correct way, sometimes I

			feel I haven't set the right word in the right location. Or more synonyms in hand to find some more complex words then the ones that I already know."

Table 1: Qualitative Result Summary

Instructor Results

The author also made several qualitative observations after acting as an instructor during the field study. The author noticed that beyond offering students assistance, the tool also has the potential to increase student awareness. Whereas while programming, students are typically aware they are facing difficulty because their program does not run or compile, when students are writing a document, they can be oblivious to the fact that they are writing poorly. This lack of awareness increases the value of this tool – it can serve not only to help teachers focus on struggling students, but also to make students aware of their own difficulty.

In addition, the author found that the previous research on self-reporting [17] was confirmed. Even when the author/instructor entered into the document environment to offer help to a student, the student was more likely to hit the "Facing Difficulty" button than ask the instructor a question directly in chat. This supports the hypothesis that students are often too shy to ask a human for help and consequently under-report difficulties. This result may also be because students felt intimidated asking for help from an instructor and found engaging with the button that was purposefully presented as a "peer companion" [26] more desirable.

Finally, the author observed that it was difficult to manage and respond to even just six students writing at the same time. A future solution to this could involve contacting different instructors depending on the difficulty being faced. For instance, the professor could be contacted for content-related difficulties, the reference librarian could be contacted for source-related difficulties, and the writing center could be contacted for grammar-related difficulties.

This could also be improved if students completed their assignments at more distributed time intervals; all of the six students in this study completed the assignment within the same day.

Quantitative Results

Table 2 contains a quantitative summary of the results from the study to supplement the qualitative evaluations. The first row shows the number of times the algorithm predicted difficulty. Clearly, it predicted difficulty very infrequently. This issue will be addressed in further detail in the analysis. The second row shows the number of times students manually indicated they were facing difficulty by pressing the button. The third row shows the total number of predictions that were made by the algorithm for each student, and the fourth row shows the number of times that the author offered help to the student.

	S1	S2	S3	S4	S5	S6
Predicted difficulties	0	0	0	1	0	0
Manually Indicated difficulties	4	0	0	1	0	8
Total predictions made	202	131	219	123	69	268
Help offered	4+ times	0 times	2 times	1 time	1 time	8+ times

Table 2: Quantitative Result Summary

Analysis

Lack of Difficulty Predicitons

The lack of difficulty predictions mentioned in the Results section was likely due to several factors. First, a two-page paper is a far shorter and less command-rich task than a typical programming assignment. As mentioned earlier, the author tried to account for this when making

initial adjustments to prediction parameters such as startup lag and segment length, but the field study revealed that further adjustments need to be made.

In addition, several aspects of the new command mapping that was created may have contributed to the lack of difficulty predictions. First, Debug commands are a strong indicator in EclipseHelper that a student is facing difficulty. Many of the commands that the author mapped to the Debug category were related to the revision stage of the writing process. However, most of the students who participated in the study did not engage in a lot of revision. This may be a writing skill that they have not learned or one that they did not care to use during such a short writing assignment, and this missing step likely contributed to the lack of difficulty predictions. Interestingly, these results are similar to EclipseHelper creator Carter's discovery that more seasoned developers use the debug tool more often than programmers with less experience [24].

Second, as seen in Figure 6, the author chose a command mapping that mapped Insert and Delete commands to the Edit category, as this was the mapping Carter and Dewan used in their field experiment. There is an alternative command mapping that maps Insert and Delete commands to their own respective categories and gives better results in laboratory experiments [24], and this alternative mapping may have generated more difficulty predictions.

Sequential and location-based command analysis might also improve results. Currently, EclipseHelper only supports difficulty predictions based on aggregate command ratios. The ability to look at location-based sequences of commands would allow for difficulty predictions to be generated when a student spends a long period of time editing the same section of a document, or when the student scrolls to a new part of the document to make a few edits.

Finally, viewing playback of the students' papers also revealed that periods of inactivity often serve as indicators of difficulty. Students indicated this in their qualitative feedback as

well. This was not the case in the original EclipseHelper algorithm, which assumed that periods of inactivity corresponded to students working on an unrelated task. However, the field study revealed that most students completed their paper in one session. This finding might differ depending on the length of the writing assignment.

Future Improvements

There are a number of areas for this project to go in the future. In addition to the potential improvements mentioned earlier, this project would benefit from more rigorous statistical analysis on the data collected in this study. Statistical analysis using the Weka tool [30] has helped to improve the EclipseHelper algorithm, and it would be beneficial to conduct a similar analysis on the results from this study.

Another area for future exploration is enhancing the teacher's user-experience. At this point, the teacher is simply sent an email indicating that a student is facing difficulty. An interface that would allow teachers to view notifications from all students in one place, view details on a specific student, and visualize statistics on the assignment level should be valuable.

Conclusion

There are several key contributions that have been made by this thesis. First, the author had the initial idea of implementing a mixed focus system for document writing, which previously did not exist. This architecture was then built and implemented. Finally, the success of this new architecture was evaluated via a field study. The field study revealed that students find feedback received through the tool helpful, but it also revealed that there are differences between writing and programming that require modifications to be made to the difficulty inference algorithm.

Figure 7: Visualizer tool developed by author

Even more significant than these results is that now that an initial set of user data has been collected, there is a testbed that can be used to re-run experiments on this data set. Using the testbed, researchers can try out new command mappings, conduct new experiments without the overhead of a field study, and train new algorithms. This testbed includes a tool that the author has built that allows researchers to visualize the command ratios used to make predictions within EclipseHelper [Figure 7, 31]. This allows researchers to easily identify what caused the algorithm to make incorrect predictions. This thesis, in combination with the visualizer created by the author and tools that have been developed by others, provides the basis for the future development of a writing-specific difficulty inference algorithm.

Works Cited

1. Bjork and Raisansen 1999; Galbraith 1999; Hayes and Flower 1986, Scardamalia and Bereiter 1991, Tynjala 2001, cited in Lindblom-Ylanne, Sari, and Heikki Pihlajamaki. "Can a Collaborative Network Environment Enhance Essay-writing Processes?" *British Journal of Educational Technology* 34.1 (2003): 17-30. Web.
2. Magin, Douglas, and Phil Helmore. "Peer and Teacher Assessments of Oral Presentation Skills: How Reliable Are They?" *Studies in Higher Education* 26.3 (2001): 287-98. Web.
3. Ertmer, Peggy A., Jennifer C. Richardson, Brian Belland, Denise Camin, Patrick Connolly, Glen Coulthard, Kimfong Lei, and Christopher Mong. "Using Peer Feedback to Enhance the Quality of Student Online Postings: An Exploratory Study." *Journal of Computer-Mediated Communication* 12.2 (2007): 412-33. Web.
4. Strijbos, Jan-Willem, Susanne Narciss, and Katrin Dünnebier. "Peer Feedback Content and Sender's Competence Level in Academic Writing Revision Tasks: Are They Critical for Feedback Perceptions and Efficiency?" *Learning and Instruction* 20.4 (2010): 291-303. Web.
5. Gielen, Sarah, Lies Tops, Filip Dochy, Patrick Onghena, and Stijn Smeets. "A Comparative Study of Peer and Teacher Feedback and of Various Peer Feedback Forms in a Secondary School Writing Curriculum." *British Educational Research Journal* 36.1 (2010): 143-62. Web.
6. Hanjani, Alireza Memari, and Li Li. "Exploring L2 Writers' Collaborative Revision Interactions and Their Writing Performance." *System* 44 (2014): 101-14. Web.
7. Wigglesworth, G. and N. Storch, *Pair versus individual writing: Effects on fluency, complexity and accuracy.* Language Testing 2009. 26(3): p. 445-466.
8. Biria, R. and S. Jafari, *The Impact of Collaborative Writing on the Writing Fluency of Iranian EFL Learners.* Journal of Language Teaching and Research, 2013. 4(1): p. 164-175.
9. Azevedo, R., D.C. Moos, J.A. Greene, F.I. Winters, and J.C. Cromley, *Why is externally-facilitated regulated learning more effective than self-regulated learning with hypermedia?* Educational Technology Research and Development, 2008. 56(1): p. 45-72.
10. "Grammarly: Instant Grammar Check, Plagiarism Checker, Online Proofreader." *Free Grammar Checker*. Grammarly Inc. Web. 17 Apr. 2016.
11. Nadasdi, Terry, and Stefan Sinclair. "SpellCheckPlus: Online Spelling and Grammar Checker for English as a Second Language." *SpellCheckPlus.* Nadaclair Language Technologies, 2016. Web. 17 Apr. 2016.
12. "The CyWrite Project." *The CyWrite Project*. Iowa State University, 2016. Web. 17 Apr. 2016.
13. "SAGrader: Help Your Students Learn through Revision." *SAGrader*. The Idea Works, Inc., 2013. Web. 17 Apr. 2016.
14. "NCWrite: Preparing Next Generation Writers." *NCWrite*. Measurement Incorporated, 2016. Web. 17 Apr. 2016.
15. Zyto, S., D.R. Karger, M.S. Ackerman, and S. Mahajan. *Successful Classroom Deployment of a Social Document Annotation System*. in *Proc. CHI*. 2012. ACM.; Yoon, Y. and B.A. Myers. *Capturing and analyzing low-level events from the code editor*. in *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools*. 2011. New York.

16. Yoon, D., N. Chen, B. Randles, A. Cheatle, S. Jackson, C. Loeckenhoff, A. Sellen, and F. Guimbretière, *RichReview++: Deployment of a Collaborative Multi-Modal Annotation System for Instructor Feedback and Peer Discussion*, in *Proc. CSCW*. 2016.
17. Johannesen, T., and E.M. Eide. "The Role of the Teacher in the Age of Technology: Will the Role Change with Use of Information and Communication Technology in Education?" *European Journal of Open and Distance Learning* (2000). Web.
18. Herbsleb, J.D., A. Mockus, T.A. Finholt, and R.E. Grinter. *Distance, dependencies, and delay in a global collaboration*. in *Proc. CSCW*. 2000.
19. Begel, A. and B. Simon. *Novice software developers, all over again*. in *International Computing Education Research Workshop*. 2008.
20. LaToza, T.D., G. Venolia, and R. Deline. *Maintaining mental models: a study of developer work habits*. in *Proc. ICSE*. 2006. IEEE.
21. "Google Docs: About." *Google Docs*. Google. Web. 17 Apr. 2016.
22. Anderson-Inman, Lynne, Carolyn Knox-Quinn, and Peter Tromba. "Synchronous Writing Environments: Real-time Interaction in Cyberspace". *Journal of Adolescent & Adult Literacy* 40.2 (1996): 134–138. Web.
23. Dewan, Prasun, Puneet Agarwal, Gautam Shroff, and Rajesh Hegde. "Mixed-focus Collaboration without Compromising Individual or Group Work." *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '10* (2010). Web.
24. Carter, Jason, and Prasun Dewan. "Mining Programming Activity to Promote Help." *ECSCW 2015: Proceedings of the 14th European Conference on Computer Supported Cooperative Work, 19-23 September 2015, Oslo, Norway* (2015): 23-42. Web.
25. Carter, J. (2014). *Automatic difficulty detection* (Order No. 3633947). Available from Dissertations & Theses @ University of North Carolina at Chapel Hill; ProQuest Dissertations & Theses Global. (1612449483). Retrieved from http://libproxy.lib.unc.edu/login?url=http://search.proquest.com/docview/1612449483?accountid=14244
26. Kapoor, Ashish, Winslow Burleson, and Rosalind W. Picard. "Automatic Prediction of Frustration." *International Journal of Human-Computer Studies* 65.8 (2007): 724-36. Web.
27. Somers, James. "How I Reverse Engineered Google Docs To Play Back Any Document's Keystrokes." *James Somers*. 4 Nov. 2014. Web. 17 Apr. 2016.
28. Hayes J R and Flower L S (1986) Writing Research and the writer. *American Psychologist* 41 1106 – 1113.
29. Olson, J.S., J. Grudin, and E. Horvitz. *Toward Understanding Preferences for Sharing and Privacy*. in *Proc. CHI*. 2005.
30. "Weka 3: Data Mining Software in Java." *Weka 3*. The University of Wakato, n.d. Web. 18 Apr. 2016.
31. Long, Duri, Nicholas Dillon, Kun Wang, Jason Carter, and Prasun Dewan. "Interactive Control and Visualization of Difficulty Inferences from User-Interface Commands." *Proceedings of the 20th International Conference on Intelligent User Interfaces Companion - IUI Companion '15* (2015). Web.