**Chapter 7**

**Future Works**

The research that we have currently completed is merely the tip of the iceberg. There are currently four primary fields that we believe need to be handled in the future to extend the power and validity of this research. The first subject that we would like to see extended upon is handling specific coding styles such as Hungarian Notation, vertical alignment, Whitesmiths style, and indentation styles. The second focus of our future research is to extend the number of languages that our method works with. The third focus of our future research is to handle the various levels of coding skill as well as bad coding practices that are in use today. The final focus of our future research is to be able to search merge history within version control to identify exactly when and by whom code has been commented out.

When dealing with unique and specific coding styles such as Hungarian Notation, vertical alignment, Whitesmiths style, and various indentation styles there are many different problems that must be considered. As discussed in further depth in the threats to validity section Hungarian Notation and Whitesmiths style cause different values that would not normally indicate commented out code to indicate commented out code. in these cases, it needs to be decided if unique trees need to be created for these problems or if there is a way to incorporate these styles into the current tree without causing trouble for the more common coding styles. Vertical alignment and various indentation styles that involve large amounts of spacing also provide a unique problem as the incorporation of excessive spaces can skew our values, again this is a problem that will either warrant the creation of unique trees for lines with excessive spacing or a way to modify the values without creating bias when integrating them into the tree. There is some argument that if they are not included in the tree creation however, that they will likely be properly identified in either case, though this is something that will require further analysis.

With the future of srcML set to expansion of their version one of this fantastic parsing tool there is a lot of hope that we will be able to collaborate in order to help us gain access to a wider suite of languages to work with. To this end work is under way to develop a new and unique method of adding parsing capability to srcML, this comes in the form of developing a system that can be fed grammar rules of the languages that you want to have parsed and have srcML auto-generate a parser for the language.

In the chapter of threats to validity it is discussed that the scope of this study is limited, because it has only worked with twenty very popular projects from Github. Now of course, this choice was made with the idea in mind that we wanted to have a very well written sample of code to work with for the first iteration of this project and it did give us access to almost 100,000 lines of comments. However the code in these projects tend to be very well written and fairly uniform, and while this does give us a good example of what code and comments should look like it does not account for junior programmers who are still learning the ropes and veteran programmers who use out of date coding styles. Of course, a third group of coders, those who are self-taught, and who lack common and good practices and standards within our field also provide an additional layer of content that we wish to explore. When looking at these groups of programmers and their coding styles they have to potential to cause shifts in the data similar to the highly specific coding styles discussed earlier in this chapter. However, the difficulty here is that unlike with those coding styles which have established rules within their designs, the coding styles that we are talking about here are much harder to identify and will require a lot of research to automate their identification.

Finally, a big part of our future research, and one of the purposes that this research is a major steppingstone for is the ability to automate the process of locating exactly when commented out code has been introduced to a script. Once we can identify when commented out code has been added into a script then we can also figure out who actually commented out the code in the script. This allows us to ask the programmer exactly they commented out the code in the first place hopefully find good solutions to the removal of this commented out code so that when a project is finally shipped it will be much easier to maintain.