Writing 3 Revised

Codenser's three main functionalities (pseudocode generation, execution tracing, code base visualization) are things that for the most part have already been attempted. Execution tracing is a very common debugging tool and plenty of technologies already exist for doing so. We will be using something called the JVM Tool Interface to pull the execution tracing data from java programs that are running, but plenty of already developed products do something similar. Many applications have accomplished execution tracing before. CodeSonar is one such example, but most of the execution tracing tools also have some sort of visualization integration. And the application Pseudogen has already accomplished pseudocode generation, although not in the way we are aiming for. Libraries like Spoon give tons of information about Java applications, and summarization/translation are classic natural language processing problems that have been accomplished many times over.

What makes Codenser novel is not the technologies that go into it, but rather the integration of all the pieces together. While each problem has been solved individually and products exist for those solutions, all three of these things complement each other in such a way that they really belong together in one application. The best way to understand new code bases quickly is to use visualization of code with execution traces and succinct summaries of what the code is doing. No other application exists that does all three of these things: visualization, execution tracing, and summarization. The one technology going into our product that is unique and innovative is the pseudocode generation. As previously stated, there does exist an application called Pseudogen that generates natural language translations of machine code. It does so by using machine learning models, a really creative application of machine translation technology. But the text Pseudogen generates is literally a line for line translation of the code; it does not make it any shorter or easier to understand. In some situations, it may actually increase the time a developer would take to read and understand what the code does. Our goal at Codenser is to make the learning process as smooth as possible. Using a combination of machine learning from Pseudogen and the code analysis from Spoon, our pseudocode will be a summarization of what the code does, not a direct translation. This functionality, integrated with the execution tracing and code visualization, will make Codenser the most effective tool on the market for onboarding new developers, and it is this integration that separates Codenser from the already existing tools.

Technical feasibility

For Codenser, we are going to use different existing tools in order to complete the functionalities it is designed for. Codenser has three major functionalities: first of all, it can take in the source code and generate pseudo code, which is easier and faster for developers to read. Although it can make group projects more efficient, the difficulty of fully developing it is challenging mainly because of two factors. The first one is the analysis process of the source code. A program can be complicated based on how it is structured, how many classes does the source code had and what are the relationships between different classes and so on, therefore, it is hard to analyze all of these by a single program.We have an existing tool called spoon to

solve all the problems mentioned before. “Spoon is an open-source library that enables programmers to transform and analyze Java source code. Spoon also allows us to parsing the data. In addition, Spoon can takes as input source code and produces transformed source code ready to be compiled” (spoon. java) which solves the problems we are facing. ” Although there are more organizing work needs to be done, we are confident that with the help of Spoon, we can analyze the source code successfully. The second challenge for the first functionality Codenser has is the generation of pseudo code. There is also an existing tool we can use to solve this, and the program is called Pseudogen. Pseudogen is a program that can analyze the source code and generate the pseudo code which is similar to Codenser, however, the pseudo it generates is redundant and it is even longer than the original code. The purpose of Pseudogen is obvious different than ours, however, we can use the method of how Pesudogen generate the pseudo code and simplify based on that. Even though more algorithms are needed to develop the simplification process, we believe it is doable by organizing the result Pseudogen provides and only present the essential part of the source code to users.

Building on the pseudocode generation functionality, the second functionality that Codenser has is an execution trace of the entire source code, which is designed to show the order in which the functions are running. Codenser iis helpful because it explains the logic behind the source code and make it easier for users to understand. A tool called the JVM Tool Interface is helpful for this specific functionality. “JVM is known as Java Virtual Machine which acts as a run-time engine to run Java applications. JVM is the one that actually calls the main method present in a java code. JVM is a part of JRE”(Java Runtime Environment). It can also develop Java code on one system and can expect it to run on any other Java enabled system without any adjustment. With the help of JVM, it is realistic to make an execution trace. Last but not least, Codenser can generate a graph which gathers all the information, users are able to click into different branches to see the details about that specific part of code. For this part, there are many tools that can be helpful, we decided to use the java library tools which can generate graph based on the code. Therefore, once we are able to generate pseudo code, a visualization of the entire source code is completely doable. In all, we have all the tools we need to develop Codenser, and we are confident that it will fulfill all the functionalities we designed at the first place.

Cost, risks, and risk mitigation

The hardware and software costs for this project are around 5,000 dollars, a very small amount compared to the impact and potential profit it can generate since most of our tools are open source so we do not have to pay for them. This means that we will be finding a way to make our product different than other similar tools currently on the market (Like Pseudogen) by improving upon and combining already existing functionalities from these tools. Furthermore a risk that this project takes is competing against these tools which are free and already provide a decent amount of functionality. We plan to mitigate this risk by providing multiple layers of functionality: pseudocode, visualization, and execution tracing along with an attractive and easy to understand user interface which will demonstrate the superiority of our product. This will make Codedenser the best option in its market which will attract more users who will in turn

recommend it to their peers bringing more exposure. Being better and providing more functionality than the competition will mean having a larger and more complicated codebase than said competition.

Codedenser will have a larger codebase than the competition but it will still run in a short amount of time since it will take around 5000 to 6000 lines of code given that it will provide more than just one functionality with each functionality taking a significant amount of lines to function properly. For example, the visualization aspect needs to get all the java classes there are, determine which one is the most important, and then using that information to compute the location of each class representation on the screen and the size it will have. All of that will require a large amount of lines and it is just for one layer of functionality that Codedenser will provide. The other functionalities mentioned above will also require many lines which will add to the total amount of lines that Codedenser will have. In addition to each component, we will have to write additional lines of code to make sure that all of these components work well together to provide a functional and polished product.

To do this we will work on separate aspects of Codedenser at the same time by having each of the members of our team work on a different one and then we will integrate all of our different parts to finish our product. Some milestones for our project are: finding tools that we will use to develop our product, testing the tools that we found by having simpler examples than our final target, scaling up from simple examples to more complex ones, and integrating different parts of the project together. We have already found tools that will be used for the development of Codedenser and we are currently working on having simple examples of these tools performing their job in our product's context. We estimate that by the end of this year we will reach our tool testing milestone and within the first half of next year we will have both large scale examples and integration completed.

With integration completed our product will be ready to launch into the market and we believe that Codedenser will take the lead against other competitors in the market and will be an extremely useful tool for consumers. This means that it will be successful overall and will bring innovation into its target market.