Writing 1 Revised
Jiawei Zhang

Programmers work more frequently with each other nowadays, and it is inevitable for them to understand the code from other members in order to integrate their work and connect every component together. In many software companies, a project can be too much work for a single developer, that is why in most cases, it is divided within a group. During this process, it is not realistic for other members to walk you through their work, considering the large workload of the project in companies, therefore, developers need to read the code by themselves and try to understand them. For a company such as Google or Facebook, the workload of a project is usually magnificent, sometimes it can be millions of lines of code. Under this circumstance, it is not only time consuming considering the size of the project, but also a waste of time if the code from other members is redundant. I believe everyone has run into challenges while working with code from others, and it can be really frustrating. But imagine there is a program that a developer can run the code base on, and it will return the pseudocode as well as the simple description for the source code. This would no doubt, make the code base much easier to read and therefore accelerate the process of working as a team. In fact, this is exactly the project we are working on. Beside the functionality mentioned above, this program named “Codenser” can also provide an execution trace for developers to get a better understanding of the source code which is really helpful when trying to understand the logic behind the code. Last but not least, the program can present a visualization of the entire source code by organizing it into an intuitive diagram.

This program is called Codenser, and we believe it will make a great contribution for the Computer Science industry for the following reasons. First of all, it can save a remarkable amount of time for every developer as well as the entire team. Take one of the classes in our school as an example: we had an operating system project in junior year, and the goal was to build a file system server. Although each of us had a clear goal on the part that we were responsible for, we had a really hard time integrating each component together because the code had thousands of lines. In the end we had to explain to other group members about the part that each of us was working on and that was extremely time consuming. This was just an experience of working on a project in school, there is no doubt that projects in industry will be much harder and more complicated than those in college, therefore, time wasting would be a much bigger issue that no one can ignore any more. Based on research conducted on team projects, the average time developers spent on integration take up to almost 40 percent. This is when Codenser can be extremely useful, by giving a simple description of the source code, developers can save most of the time reading the code from others. In addition, it can provide information on what functions are running in what order by showing the execution of trace. With these functionalities, Codenser can greatly increase the project efficiency and make it more convenient for developers. This is why this program is worth founding, it will make more convenient and efficient for programmers when they are working with others.

However, there will be some technical challenges when we develop Codenser. For example, summarizing the source code to pseudocode is already very difficult, not to mention our final goal is to give a brief description in English sentences. In addition, can the execution trace provide valuable information when there are hundred of functions? Last but not least, what if the source code has bugs? Can Codenser still give useful summary of the code? Questions like these make the development process really challenging, but we believe that our team will solve these problems and Codenser will be a success.

Writing 1 Revised

Braden Meyerhoefer

The technology that is a large part of everyone's daily lives runs on massive code bases. Popular applications such as Microsoft Windows have millions of lines of code and require constant updating to maintain security, relevance, and functionality. This is a long and hard process, especially when releasing broken code or failing to make meaningful updates and additions to products in a reasonable time period can lead to huge losses of revenue. This process is even harder for new developers, who must first read and understand the existing code before they can start producing changes to it. Right now, the main way of learning how code works is by reading comments, documentation, and the code itself. But comments can be unreliable, and documentation hard to read and understand. Developers need a tool for easily understanding huge, hard to comprehend code bases. And right now, there isn't one. Our product, Codenser, will be able to fill this void.

As a useful metaphor, imagine a car mechanic who has been in the workforce for a couple years. This mechanic has studied some sort of engineering in school and worked with a number of different cars fixing and possibly modifying them; maybe they've even built their own car from scratch, engine and all. It is safe to say that this mechanic has a pretty in-depth knowledge of how engines in general work, basic electronics and hydraulics, all the things that go into building a car. Now that mechanic decides to switch gears and joins NASA, working on a team that is building a brand-new spaceship. To help get them up to speed, the team provides them with a complete schematic of the rocket, complete with a 10000-page booklet of documentation explaining everything. This is kind of like what it is to be a programmer joining a new team. All developers have a basic understanding of how code works, how computers work, how applications work, etc. But this does not mean that they can jump into a massive new project and immediately understand how everything works and make changes to it. Even the best rocket scientist in the world can't just look at a new rocket and immediately start improving or fixing it. But what if instead of reading a ridiculous amount of mind-numbing documentation, the mechanic could press a button on the rocket and see a diagram light up highlighting exactly which components are being activated, how they work together, and get a summary description of what each part does. That would make familiarizing themselves with how everything works on the new project so much easier.

This is what Codenser aims to do with code. A new developer can run the code base through the application, which will return to them pseudocode, execution traces for all functionality, and organize it all into an intuitive, interactive diagram. The developer could then click on the various functionalities of their project and see exactly which classes are being used, which functions are running and in what order, and get pseudocode descriptions of what these functions are doing. There will be technical challenges of course. Writing good pseudocode can already be difficult, and automating that so a computer can read code and deduce what is important and what it is trying to do is even harder. Execution traces can be technical, long, and hard to follow. Portraying or summarizing that in a meaningful way will be a challenge. And usefully conveying all of this information in a visually pleasing and efficient way does not have an obvious solution either. These are all hurdles that we as a team believe can be overcome and are seeking funding so that Codenser can become an application assisting development teams everywhere.

David Alejos Revised Writing 1

Codedenser

Many software companies today assign projects to a team of people instead of a single person. This can be beneficial for the developers involved because the work can be evenly split up and some developers can specialize in one part of the project so efficiency can also be increased. However, for developers working in a team is more than just creating a part of a product since after everyone has done his or her part the code from the project needs to be integrated together. Problems are very likely to arise from this because there is no one person in the team who worked on everything so explaining is bound to happen between team members during integration. Integrating parts of a project together gets extremely complicated when the project is of a large size both in workers and amount of code. This also implies that members of a project will have to exchange information about the code which is another source for complications. This can often lead to a large variety of problems such as incompatibility between different project modules, unexpected bugs caused by other parts of the project, and unexpected input or output from a module due to unknown behavior from the rest of the project. Our product, Codedenser, aims to take a piece of code and then explain what it does in more understandable terms like a mixture in English and code (Also called pseudocode). Solving these and many other problems resulting from integration is no small task and that could spell major trouble for both developers and companies due to missed project deadlines caused by inability to integrate said project which can end up creating conflict between the two parties.

These problems will be avoided with the use of our product because it aims to be a tool used for prevention and solving the complications mentioned above by making it such that the idea or ideas behind code can be easily understood without having to read through all of it. This can be especially useful when a person needs to understand a piece of code if the author is not present or not available to talk to. With our product there will be no need for the author of the code because it will take in the code and then provide the ideas behind it. If the ideas behind code are laid out to be read then many integration problems for group projects would be solved because there will be a simple, reliable, and consistent explanation for all parts of the project granted by our product. This is bound to make teamwork and integration for code-based projects much easier than having team members constantly explain their code to others by saving time and by having only one universal explanation for the code. Companies will be benefited from this by being able to meet deadlines, avoid stress for developers, and avoid conflict between different parties working on the same project.

Codedenser will alleviate many problems for both companies and developers but there will be some technical difficulties in creating it. Some examples of this are giving a consistent summary of similar (or identical in function) pieces of code and summarizing said code to a shorter and more understandable form. These complications will be resolved through testing and careful design to make a successful product. Investing in our product has the chance to bring in considerable profit due to the sheer amount of applications that it can have for many different software companies.

Writing 2

Codenser at its core really does two things: it traces execution pathing and generates pseudocode summaries of methods. These are by no means new ideas and there are applications that do them already. For the Python programming language, a tool exists called Psuedogen, which generates pseudocode from Python source code using machine learning. It naturally does this for Python to English and Python to Japanese, but also provides a framework for doing this for any desired source code/natural language pairing. There is also a multitude of tools for execution tracing on the market, many of which are free. If a developer is coding in an IDE, chances are the debugger included in that application has some sort of trace function. In addition to that, tools such as Flow and HyperDebug provide execution tracing with GUI visualizations.

These tools all have been around for a while and work well, but we as a team believe that Codenser will provide significant improvements on what these tools have to offer. Starting with pseudocode generation, the pseudocode that Psuedogen produces seems to be easy to read and accurate. The problem is that it does not really shorten what needs to be read, just translates the code into natural language. That makes it easier to read, but not necessarily faster or easier to understand. It is really just reading the code for the developer, not saying what it's doing at a high level (like pseudocode should). Codenser will improve upon this by actually generating natural language summaries of the code, not just a word for word translation. Where a product like Psuedogen will take 50 lines of code and produce 50 lines of english, Codenser will take 50 lines of code and produce a couple lines summarizing the functionality.

In terms of execution tracing, the output of the current tools can sometimes be very technical and hard to read themselves. We believe the most important thing for a new developer to see is the order of execution and how classes/methods are interacting with each other, and for this reason will condense the trace down to that. Any further insight the developer wants can then be gleaned from the summarization, or inspecting the code itself when necessary. The execution trees and graphics these apps are displaying are also sometimes hard to follow as well, and we will try to make the display easier to read and follow for Codenser. And finally, where Codenser will really excel is the integration of the execution tracing and the code summarization. In this easy to follow visualization, the user will see how methods are executing and be able to click on them to not only highlight the execution path, but also view the generated pseudocode for the relevant methods. No other application has the integration of the two, and this is what will really set Codenser apart. With the advantages of Codenser mentioned, we believe it will attract many potential users who are in the Computer Science industry.

The major potential users that we are targeting are programmers who need to work on projects with others in Computer Science industry. First, because Codenser can generate a simplified pseudocode, which is a summary for the source code, developers do not need to spend a large amount of time on reading code from others. This is really helpful not only for developers, but also for companies as well. Projects in Computer Science industry usually needs million lines of code, and if one of the team members need to understand the code from a partner, reading though the code and understand sometimes needs days. However, with a program that can summarize the entire source code and provide visualization of the main branches, developers can save a great amount of time while integrating code together. In addition, it can also improve the efficiency of in-progress project for the company.

Besides professionals, students can also benefit from Codenser as well. It is common that students in school are assigned some group projects, especially in college. Therefore, it is also inevitable for them to spend a large amount of time trying to understand the code from other group members. This is extremely time consuming considering the workload of other classes. Codenser can make the group project more efficient to all of the members, so students are able to understand the source code faster. In all, Codenser will be helpful to both professional programmers who work in industry and students who are learning in school because it can make the source code easier to read and save them a remarkable amount of time.

Considering the functionalities and benefits that Codenser will bring, the project is a social-impact project which aimed at foundations. Codenser has three major functionalities, first of all, it can take in the source code and generate a simplified pseudocode as well as a brief summary in English sentences. Secondly, it can provide information on what functions are running in what order by showing the execution of trace. This functionality can help users understand the logic behind the source code, so they can have a better understanding on the original code base. Last but not least, it provides a graph of the entire source code and breaks it down into different branches with details, which makes the code from other members in the group easier to read. All of these three functionalities mentioned above solve one of the biggest problems programmers have while working in teams, which is time consuming. Based on research conducted on team projects, the average time developers spent on integration take up to almost 40 percent. If a program can help our targeted audience to save 40 percent of its total working time, they are very likely to use the product. As the developers of Codenser, we want to show our audience what can this product do and how does it can make reading code easier for them when working in groups. In addition, we want to use social media to let more people know about our product, so more potential users will notice it. By explaining the functionalities of Codenser, more and more programmers will be willing to use it. These functionalities will also attract companies that assign projects to a team of people as well as companies that have a large code base that needs to be explained to new recruits.

In today's world more companies are developing large software projects that are assigned to a team of developers instead of giving a single developer a standalone task to be completed individually. Working in a team means that developers will have to make all of their code compatible and fully functional with the rest of the code. This can often lead to a large variety of problems such as incompatibility between different project modules, unexpected bugs from integration, and unexpected input or output from a module due to unknown behavior from the rest of the project. Our product, Codedenser, aims to take a piece of code and then explain what it does in more understandable terms like a mixture in English and code (Also called pseudocode). Codedenser has the potential to greatly decrease the amount of time that it would take a team of developers to understand the entire project that they collectively worked on. This will also have the benefit of having a uniform explanation for all of the code which leaves less room for misunderstandings within the team. These two benefits have the potential to greatly reduce the time it takes for software companies to develop new software which could mean an increase in efficiency. An increase in efficiency for team projects could affect society today because more content could be developed at a faster rate than before. However, team projects is just one area where our product could change the landscape, another example would be the time it takes to train a new employee in some companies where there is already a large existing code base.

For many companies that already have developed software which has many lines of code, training an employee could be very hard for both the company and the new employee. This can be hard because for the new developer to begin working, he or she will have to read for most of the code (Possibly all of it) to begin actually working for the company. Reading through and understanding code takes an extremely long amount of time and a great deal of effort which makes the training of a new recruit expensive for a company. Codedenser will automate the understanding part of the process which will significantly reduce the time it would take to train a new employee. Reduction in training time could be an incentive for companies to hire more people since the cost of hiring a new worker would decrease since less time is used for training.

Less costly recruitment and more effective teamwork opens up the possibility for much faster deployment of new software products which will make Codedenser a very valuable tool. It will also be a very powerful tool that will have many applications but it will not need to be regulated since it will be very hard to put it to bad use. It will be hard because our product only simplifies code and many sensitive code bases are not available to the public and if it is used to help an attacker decipher an encrypted message it will not provide the attacker with an advantage. This makes Codedenser a tool that can only be used with good intentions and a strong software that many consumers will want to have.

Writing 3

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 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.

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.