CODE ANALYSER

REVIEW-3

NAME	REGESTRATION NUMBER
Anubhav Singh Guleria	18BCE0186
Modh Umar	18BCE0196
Anish Aggarwal	18BCE0281

ABSTRACT

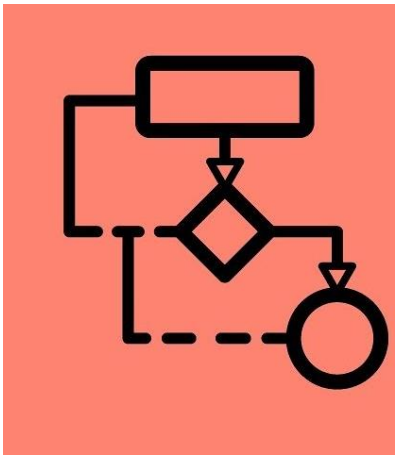
All around the world software engineers can't comprehend the fact that making charts is one of the most important tasks in software engineering which can develop an insight and help the team as a whole. Hence our project is a prototype for a program which helps builds charts such as Gantt charts, timeline charts etc. Hence we are trying to develop a basic application which can help all the software engineers around the globe. Hence our application will make basic charts and graphs in one go. Since most of the online applications do not consist of all the amenities required by the people hence with this ground breaking idea we are trying to develop a site which can be deemed as the holy grail of detailed management and idea processing. Hence with this we are trying to change the momentum of the software development. We are also trying to implement code analysis which can count the number of functions used etc. Hence with this project we can revolutionize the design and analysis phases of the software development.

INTRODUCTION

It is hard to believe that people these days study subject as their core specialization and yet don't ask the necessary questions about their existence or why were they even created.

On such subject being the space and time complexities and code optimization. A general non optimized code which takes a little more space that is could have often doesn't lead to any problem or issues in the intermediate level but as the size of the data increases so does that residual space and eventually it occupies such an amount which cannot be neglected hence this lays the foundation of optimization and this just being the tip of the ice-berg often scrutinizes and creates various subjugation on the overall computer industries. Hence what has been done for optimization? The answer is simple; better solutions , better algorithms, better approach and most importantly better testing interfaces.

Hence with this small project we would like to simply help the software engineers with a better outlook for the optimization of their codes. Has anyone ever wondered how many for loops a person's code consists of, what will be their space or time complexity what is the amount of nested loops and how many of them exits and so. So with this project we would like to do so making the life of software developers and testers a bit easier. Hence our code planar will give the user a head start over other developers and programmers so that they can observe their codes understand the complexities and work accordingly to make the results faster cheaper and with better spatial expenditure which could not on a large but a sure rate polarize the industry.

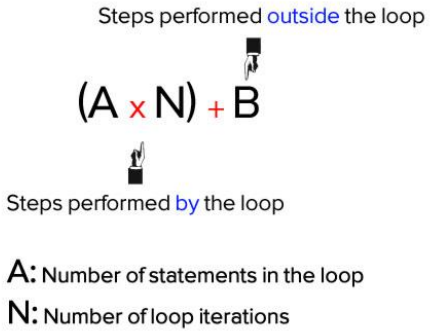


Furthermore this can be further used in many documentations with its multiple variants. Hence we can even improvise this application and enhance it using various techniques, hence this plan might actually be an amazing and adaptable application.We can even modify the application to suit the other departments. Hence the development will be done module wise so that we can further enhance a single module and not the whole project.

This application has a chance to play a pivot role in the implementation and time saving will editing, improvising, optimizing and trying newer code. Moreover our project will work as a simple yet powerful tool which could be used to develop and improvise code and still get to know what your current code will do in what time and in what space hence we are trying to overall improvise the usability

while maintaining the speed limiters and still maintaining the prejudice of the project.

This project won't code for you nor help you implement you newer simpler code. All this project will do is help you understand the maximum time complexity of your code and space complexity so that you can understand how much and what you want this tool will do that for you so that you know what does your recent code does and will help you with the calculations so that you can concentrate on the code and not the time wasting repeated concentration.



LITERATURE SURVEY

1) Debray and Lin: Automatic complexity analysis of programs has been widely studied in the context of functional languages. This paper develops a method for automatic analysis of the worse-case complexity of a large class of logic programs. The primary contribution of this paper is that it shows how to deal with nondeterminism and the generation of multiple solutions via backtracking.

2) Abhijit et al: This paper studies static complexity in manufacturing systems. This paper enumerate factors influencing static complexity, and define a static complexity measure in terms of the processing requirements of parts to be produced and machine capabilities. The measure suggested for static complexity in manufacturing systems needs only the information available from production orders and process plans. The variation in static complexity is studied with respect to part similarity, system size, and product design changes.

3) Soubhik and Suman: The present paper argues that it suffices for an algorithmic time complexity measure to be system invariant rather than system independent which means predicting from the desk.

4) C.D. Thompsan: The complexity of the Discrete Fourier Transform (DFT) is studied with respect to a new model of computation appropriate to VLSI technology. This model focuses on two key parameters, the amount of silicon area and time required to implement a DFT on a single chip. Lower bounds on area (A) and time (T) are related to the number of points (N) in the DFT: $AT2 \geq N2/16$. This inequality holds for any chip design based on any algorithm.

5) Mikhail Ju. Moshkov: The research paper is devoted to the study of bounds on time complexity in the worst case of decision trees and algorithms for decision tree construction. The results of the monograph are discussed in context of rough set theory and decision tree theory. Followed by description of some tools for decision tree investigation based on the notion of decision table general results

about time complexity of decision trees over arbitrary (finite and infinite) information systems are considered.

6) Qingfeng, Zili and M. Sha: Code size expansion of software-pipelined loops is a critical problem for DSP systems with strict code size constraint. Some ad-hoc code size reduction techniques were used to try to reduce the prologue/epilogue produced by software pipelining. We present the fundamental understanding of the relationship between code size expansion and software pipelining.

7) Kalai, Raz and Regev: In this it is confirmed whether Linear Programming remains P-complete, even if the polyhedron is a fixed polyhedron, for each input size, and only the objective function is given as input. More formally, it considers the following problem: maximize $c \cdot x$, subject to $Ax \leq b$; $x \in Rd$, where A,b are fixed in advance and only c is given as an input.

8) M.T. Jensen: This paper presents a new and efficient algorithm for nondominated sorting, which can speed up the processing time of some multi objective evolutionary algorithms (MOEAs) substantially. The new algorithm is incorporated into the nondominated sorting genetic algorithm II (NSGA-II) and reduces the overall run-time complexity of this algorithm to O(GN log/sup M-1/N).

9) Pushkar, Raghavendra and Sivakumar: This paper attempt to define a right notion of space complexity for the BSS model, using the notion of weak-space, introduced by Naurois [CiE 2007] .It is an attempt to investigate the weak-space bounded computations and their plausible relationship with the classical space bounded computations. For weak-space bounded, division-free computations over BSS machines over complex numbers with =?0 tests.

10) Carl, Srinivasan, Hank and Morris: A method of efficient code generation for modulo scheduled uncounted loops includes: assigning a given stage predicate to each instruction in each stage, including assigning a given stage predicate to each instruction in each speculative stage; and using the stage predicate to conditionally enable or disable the execution of an instruction during the prologue and epilogue execution.

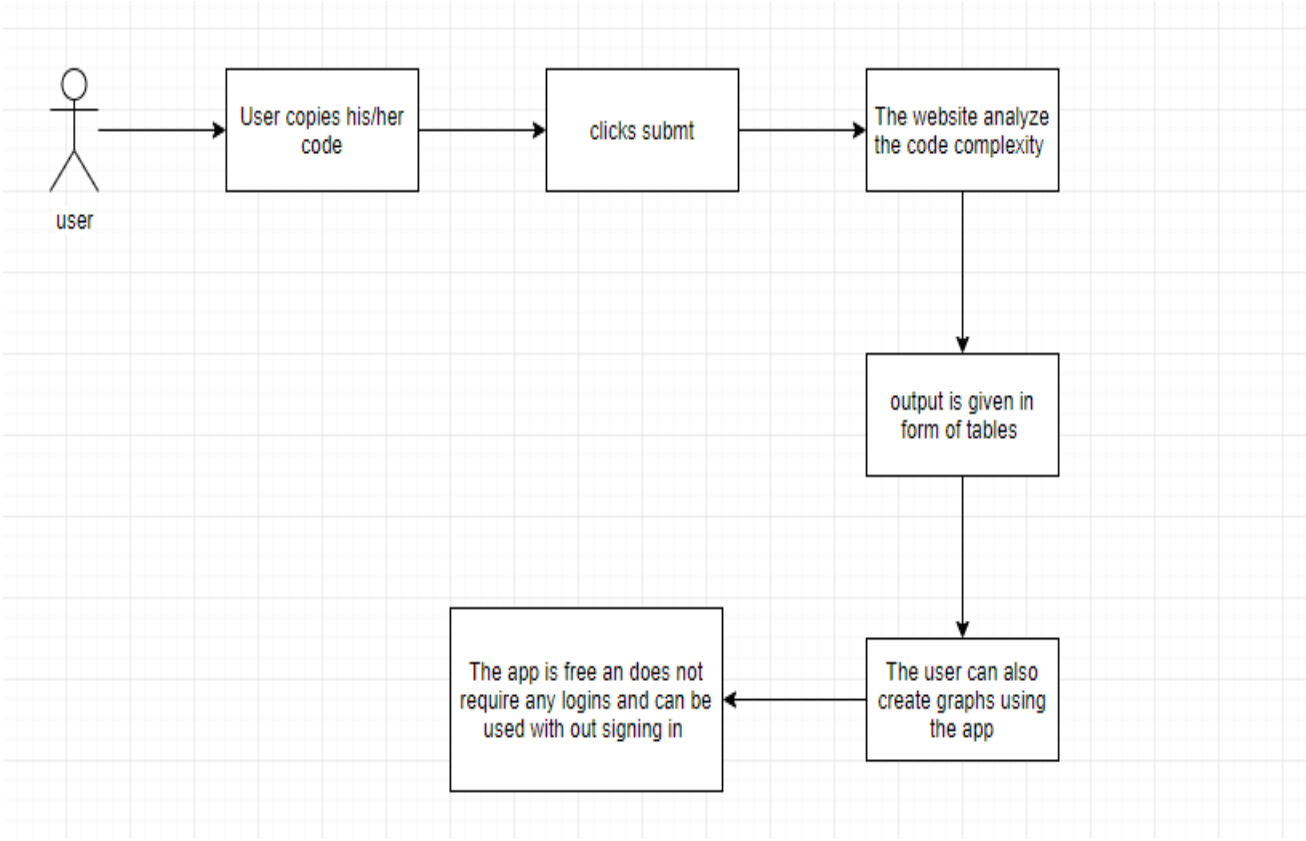
PROBLEM STATEMENT

Has anyone given this a thought that when we code the correct code is not what is important. The thing more important than it is the speed and the space that particular code would take and require. Hence with this project we would like to full fill that gap in the software prep work. No one gives this a thought unless there is some memory error or crash due to extreme time taking to complete certain process since this is the era of big data and minimalistic data age code would not be able to compete and work at the same rate as the older codes hence for that we need a better and faster understanding of the time-space complexities so as to improvise and understand faster and code better. Hence with this project we can solve that understanding issue and allow the developer faster understanding of the project so that he can further optimize and improvise the code.

METHODOLOGY:-

In our project we are implementing a code analyzer that can check the cyclomatic code complexity of the following function of the code and hence directly telling the complexity of all the functions used in the following functions used in the code. Moreover we have also provided with a complementary chart maker which helps us make charts from the csv file uploaded by the user and can also graph it. Its free to use and no login is required.

BLOCK – DIAGRAM:-



ABOUT CYCLOMATIC COMPLEXITY

We generally use the following formula. Cyclomatic complexity is a source code complexity measurement that is being correlated to a number of coding errors. It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.

Lower the Program's cyclomatic complexity, lower the risk to modify and easier to understand. It can be represented using the below formula:

Cyclomatic complexity = E - N + 2*P

where,

E = number of edges in the flow graph.

N = number of nodes in the flow graph.

P = number of nodes that have exit points

But for this code we have kept it simple and rather used a different and simpler approach which could help us code and also be diverse enough to help us find the complexity of different languages in a single simple algorithm.

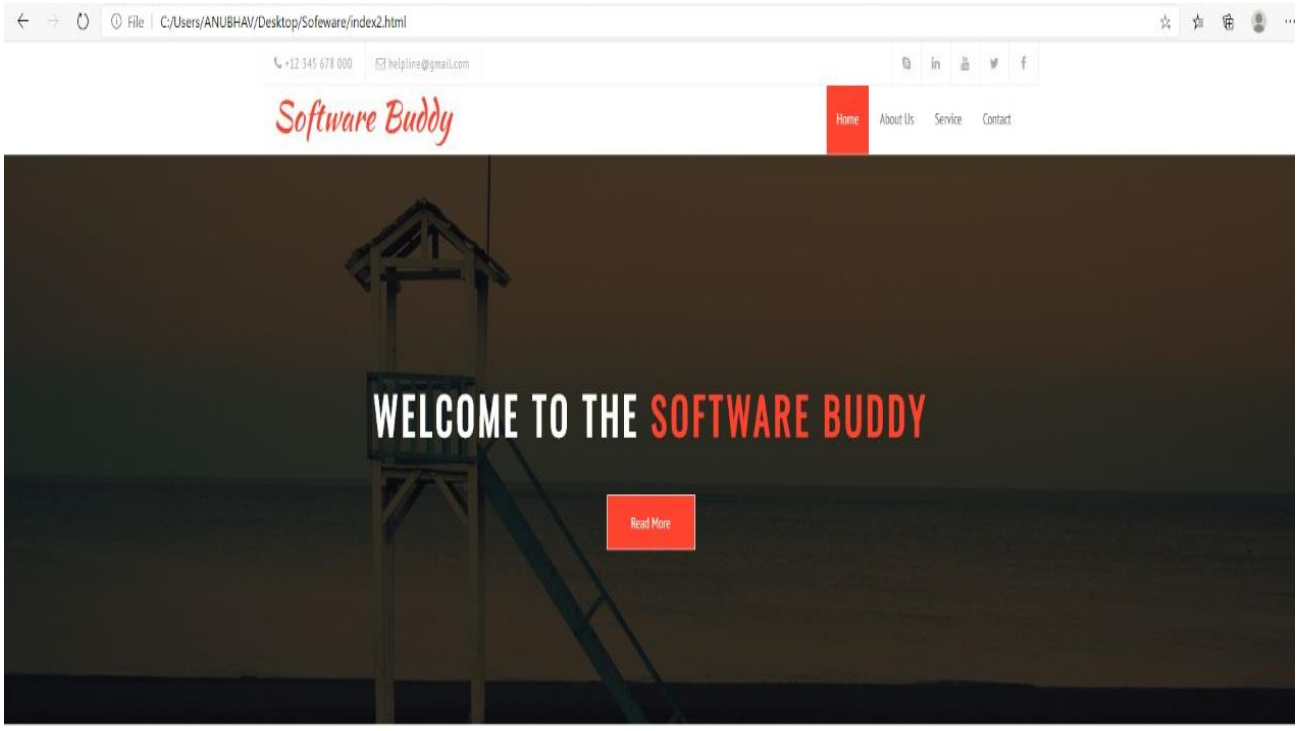
That algorithm being used here is finding the predicate node. That is complexity of the code is the number of predicate nodes +1, here the predicate nodes include the number of if , else if, switch case, for , while ,do while all these are to be included as the predicate nodes.

PSEUDOCODE:-

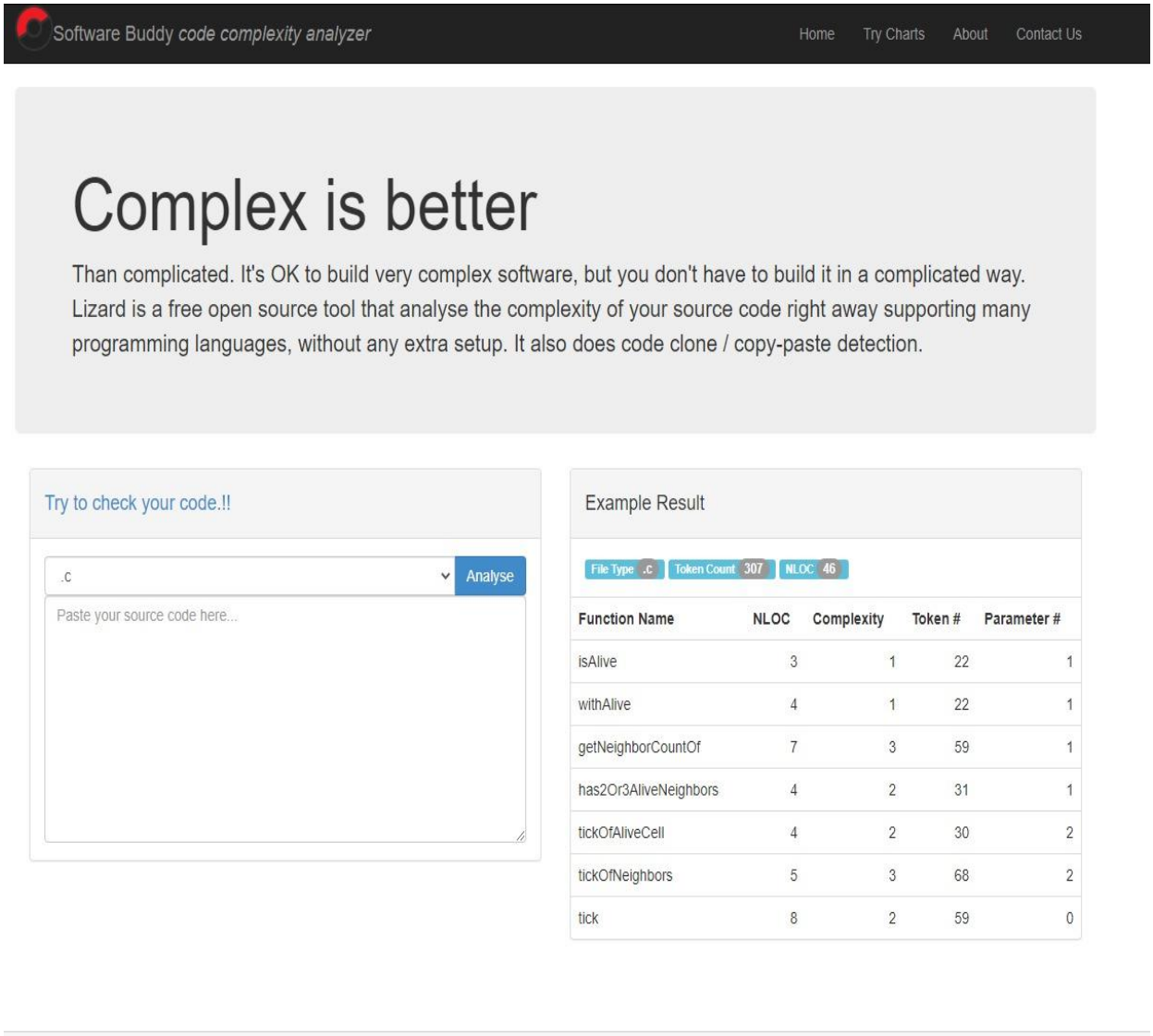
- .using html, CSS design an front end.
- .The server side could be done using the angular.js and jquery.js
- .Link the server side to the algorithm.
- .In the algorithm we will be using the regular expression using the node.js which has in build regex function to simply find the number of the nodes as explained above and simply give the complexity which can be seen in the following way.
- .We can further enhance is to it so as to understand the different function being used using the brackets and “:” .
- .And simply do repeat the whole process for the differentiated functions.

RESULTS:

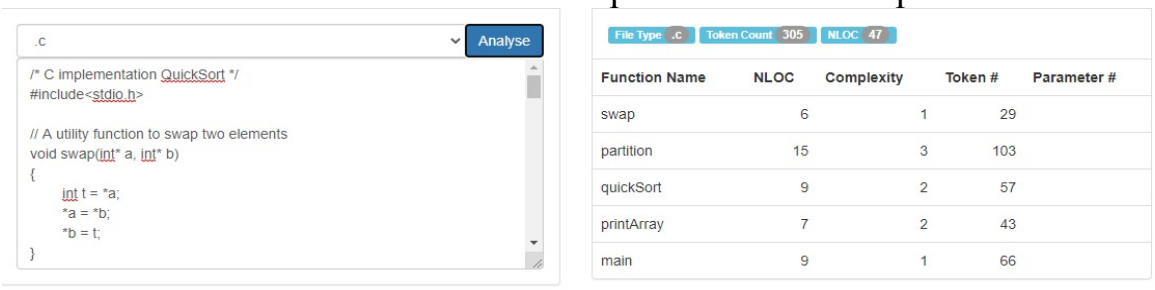
This is a basic prototype of the website SOFTWARE BUBBY, where Code Analyzer is a part of something big which is being created.



Our motto Complexity is Better to define the function of this Code Analyzer. User can paste the code of the program in the “Try to check your code.!!” and click “Analyse” to check the code’s time complexity.



This is how the working program look like, user just has to paste the code and all the statistics relevant to the code will be printed on the side panel.



Our team is also working on enhancing the website by introducing more features to it. One such feature being “CHARTS”, in which a user can upload a csv file and select from the many options a chart to make. Though this part is still being coded and not yet available to use.