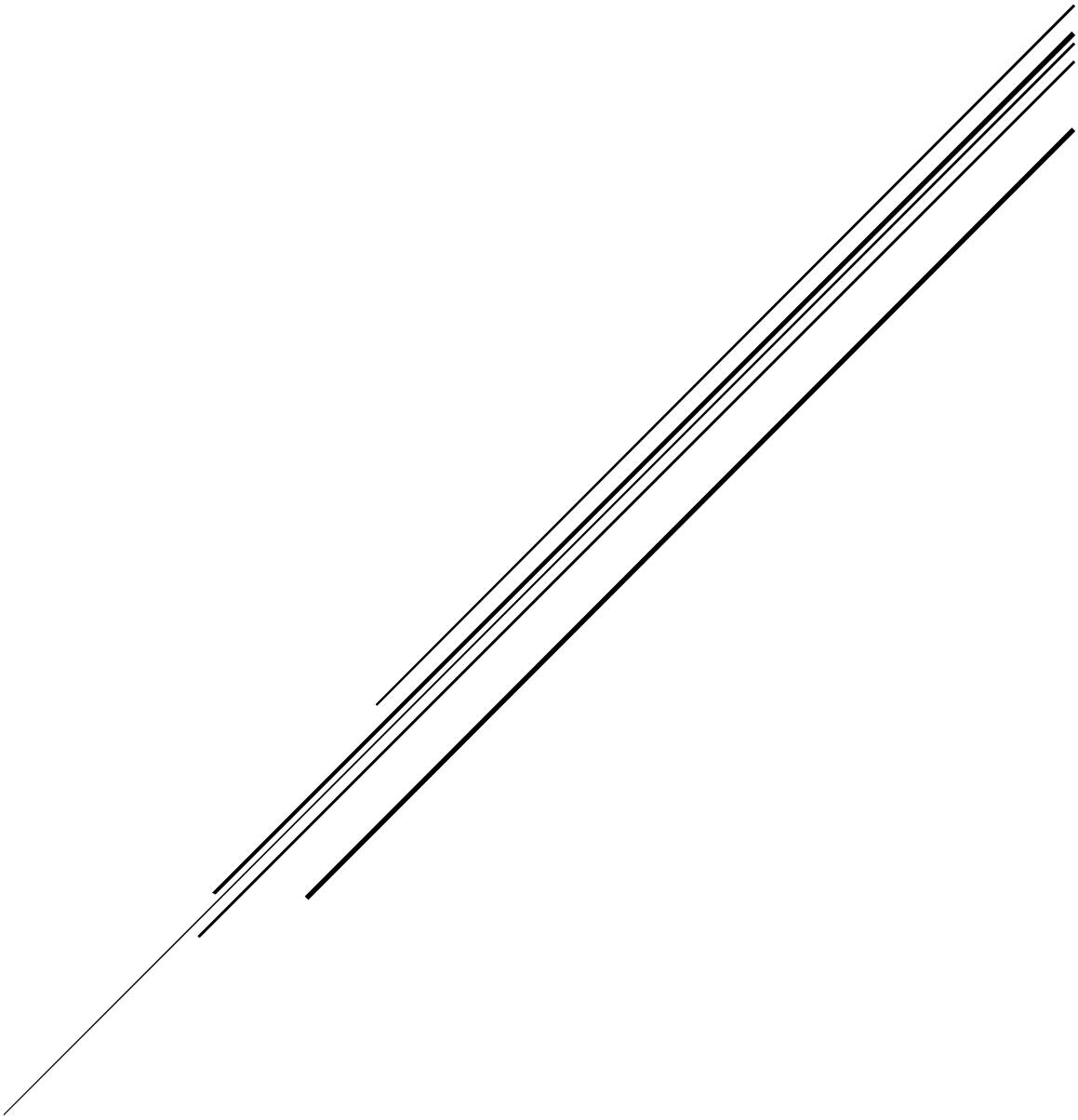
CODE ANALYZER

OPERATIONAL CONCEPT DOCUMENT

681-SOFTWARE MODELING AND ANALYSIS, PROJECT#1, FALL-2014



SUBMITTED BY: TANMAY FADNAVIS
INSTRUCTOR: DR. JIM FAWCETT

CONTENTS

1.INTRODUCTION

1.1 EXECUTIVE SUMMARY

This document presents the architecture for the Code Analyzer.

In any industry level project, a single package contains thousands of lines of code. The project contains lot of packages. Over the course of time it becomes an overhead to keep track of the structure of the code. A tool which will be used to keep track of the overall structure would be of great help for the developers, testers, code review team, etc. Code Analyzer does this work for us.

Code Analyzer will analyze the C# code. It could also be enhanced to analyze C++ code, etc. The code analyzer would be helpful for the developers in the following ways.

- It will display the various types used in the C# code i.e. classes, structs, interfaces and enums.
- It will also display the various relationships among the types i.e. inheritance, composition, aggregation and using.
- It will also display the different complexity as the number of elements in a particular scope.

The various packages in the Code Analyzer are

- Executive package.
- Command Line Parser package.
- File Manager package.
- Analyzer package.
- Parser package.
- Semi Expression package.

- Tokenizer package.
- Rules and Action package.
- Display package.
- XML processor package.

The following are the critical issues identified which will be discussed at a later stage.

- Performance of the tool due to the large number of files. The code analyzer might get slowed down due to large set of files.
- The output format to be displayed to the users. If the output contains lot of type members and relationships, also if there are a lot of complexities in a particular scope displaying in a tabular format won't be a user friendly approach.
- What if there is a logical/syntactical error in the user program. For e.g. the user opens a particular scope but does not close it.
- What if the file provided by the user is protected and cannot be opened by the analyzer.

1.2 CODE ANALYZER REQUIREMENTS/FEATURES

The code analyzer is a tool which allows the users to input their C# code file and it outputs the types, relationships and the complexities in terms of number of elements in the scope tree. The tool will provide the users will the following features.

- Shall be executed from the console.
    - The users can provide the path of the file to be analyzed along with the file pattern (.cs) from the console line.
- Shall provide an option for sub-directory search
    - The user can optionally give a command on the console itself if he wants all the sub-directories of the path to be analyzed.
- Shall provide output at the console.

  - The code analyzer will provide the output at the console itself. The output shall contain the types present in the C# code and their members. It will also display the function sizes and the complexities and scope of each functions.
- Shall provide an option to display relationships
  - The users can also view the relationships between various types found in the code by typing an extra option on the command line.
- Shall provide output in an XML file
  - The users will also have an option if they want the output to be written in an XML file instead of the console.
- Shall point at the functions which have huge size and complexities.
  - The analyzer will point to the functions which have a large size and which may involve a lot of complexities. This would help the developers and the code review team to try and optimize or redesign/recode that particular function.

Thus, all the above features will make sure that the developer will get a good understanding of the code structure and will help in designing a cleaner/optimized system.

2.POTENTIAL USERS AND THE CODE ANALYZER USES

2.1 SOFTWARE DEVELOPERS

Developers in a software project tend to produce a large number of code over the course of the total development. Code analyzer will help them to understand the overall structure of the project by displaying the various types and the relationship between them. The code analyzer could be used by developers in the below scenarios.

- Optimization of the code.

  The code analyzer will display the total size of each and every function used in the development. The developer could come to know about that function with a larger size and could possibly look for different ways to optimize the function. As large

function size could probably result in a large execution time, optimization of code is a critical step and the code analyzer could point to functions which may need it.

- The recursive path search facility will also give the developers a feature to search for all the subdirectories.

- Useful in enhancement of the product.

  If a product needs enhancement, many a times it becomes difficult to understand the code written by the developers themselves during the development phase due to the time difference between the project go live phase and the enhancement phase. Code analyzer could help in doing the same as the development team could quickly go through the various constructs that were used during development and thus will help the team in adding a new enhancement in a better and optimized way.

- Useful whenever a code change is required
  Consider a scenario where a developer realizes a bug in the code and he needs to make some changes. As the code analyzer will display the relationships between various types, the developer can quickly analyze the places where the particular code change would be necessary by just analyzing the output.

2.2 SOFTWARE TESTERS

The code analyzer would be of great aid to the testers who test the application. If the testers know the overall structure of the project, they would get a better idea of the entire application and thus could write the test codes in an efficient way. Also, if the testing team is using some tool, and if a particular test is failing, the team would be in a better position to resolve the issue as they are aware of the various types, the relationships etc.

2.3 CODE REVIEW TEAM

- Quickly understanding and helping in reducing the complexity of the code.

  By analyzing the structure of the code, the code reviewer could get a clearer picture of what is the context of the code he is reviewing. For e.g. The reviewer could look at the output of the analyzer and check if a particular scope has a lot of complexity in terms of the number of loops used, total branch conditions that have been used, etc. Thus, by analyzing that particular scope, reviewer could tell the developer to modify that particular section of code.

2.4 NEW DEVELOPERS ADDED TO THE PROJECT

If the project demands, new developers could be added to the development team. The new members could improve their productivity soon if they could understand the overall structure of the project. The code analyzer will help them in doing the same. Thus analyzer would be very helpful for knowledge transfer and reducing the learning curve of the new joiners. They could quickly learn about the various types used in the code and the relationships between them, scope of various types and thus could understand the application structure in a much efficient way.

2.5 CLIENT SIDE TECHNICAL TEAM

The code analyzer would be a very helpful tool. This could make the client's technical team understand the project in a better way and thus, could be very helpful during the handover phase.

3.APPLICATION TASKS

3.1 PROCESS USER INPUT FROM THE CONSOLE

The user will input the file path along with the file pattern and the options from the command line. The user will provide the file path which will contain the C# files which needs to be analyzed. The code analyzer will process this input along with the options if provided.

3.2 GET THE REQUIRED FILES FOR ANALYSIS

Once the user provides the required file path and the pattern, the application will be responsible for fetching the mentioned files on that path matching that particular pattern using native C# file handling mechanism.

3.3 PARSE THE FILE/FILES AS PER THE ACTION RULES MENTIONED

Once the analyzer has the required file, the analyzer will firstly pass the files and check for the types i.e. classes, structs, interfaces, enums. The analyzer will also check for the functions, their names, their sizes and the complexities. This would be done as per the rules and the actions mentioned in the code analyzer.

3.4 DISPLAY THE TYPES, COMPLEXITIES ALONG WITH THE FUNCTION NAMES TO THE CONSOLE

Once the output is ready, the analyzer will display the output on the console. It will display the various types used in the file along with the function names, their scopes, namespaces, the brace less scopes, the complexities of a particular scope i.e. the number of loops and the conditional statements used in that scope, it will also display the sizes of the functions. All this would be displayed in a row-wise format on the console.

3.5 CHECK FOR THE RELATIONSHIPS AND THE SUB-DIRECTORIES IF MENTIONED

If the user has included '/S' option on the console, the application would get all the mentioned files present in the given path's directory i.e. all the sub-directory files would also be analyzed by the analyzer. Additionally if the user provides a '/R' command line option, the analyzer would also analyze all the relationships like inheritance, using, composition and aggregation between the types and include it in the display along with the types.

3.6 WRITE THE OUTPUT IN AN XML FORMAT IF MENTIONED

The application is responsible to write the output to a file in an XML format if the user includes the '/X' option.

4.APPLICATION FLOW

The code analyzer flow would be as follows. Firstly the user would provide on the console line the file path and the file pattern whose code is needed to be analyzed. Along with this the user also has few options. He can include a '/S' option to include the subdirectories of the path.'/R' option which will analyze the relationships between the types and a '/X' option which will write the output to a file in a XML format. Thus the user input will be in the format "File->Path File->Pattern Options". Once the user gives its input, the code analyzer will parse the command line i.e. the path along with the collection of patterns and the collection of options. The analyzer will get the file references.

In the first pass, the analyzer will analyze the types in the file, save the type information for the file along with other information in the repository and then go for the next file. The types would be found based on the pre-defined rules and actions stored in the analyzer.

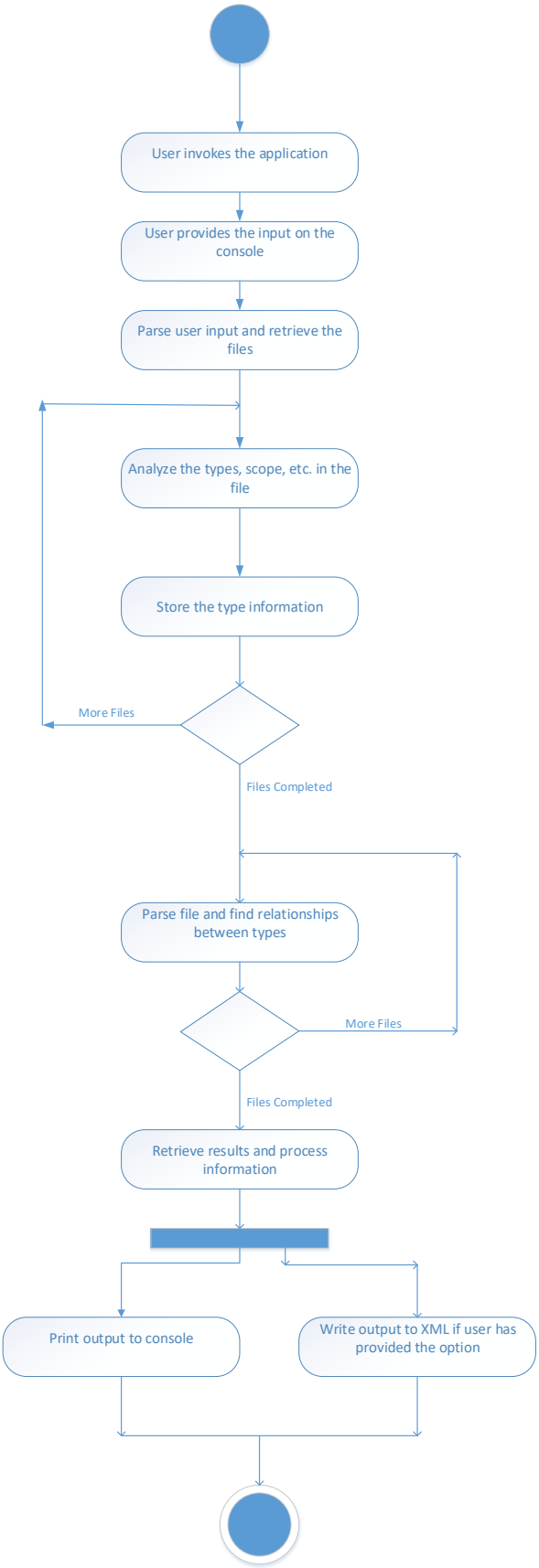
In the second pass, the relationships would be analyzed. The relationships which are pre-defined only those would be analyzed. The analyzer would find the type, examine its code for the relationship with the types found in pass 1, establish the relationship and then go for the next file.

Then, once the relationships have been established, along with all the other details, the analyzer would retrieve the results, process the information and send the output to the console for display. If the user has provided an option for an XML output, the analyzer would also write the output in a file in a XML format.

This is the overall flow of the application.

5.ACTIVITY DIAGRAM

The activity diagram acts as a high level description of the code analyzer. The below activity diagram shows logical flow of the code analyzer.


User invokes the application
User provides the input on the console
Parse user input and retrieve the files
Analyze the types, scope, etc. in the file
Store the type information
More Files
Files Completed
Parse file and find relationships between types
More Files
Files Completed
Retrieve results and process information
Print output to console
Write output to XML if user has provided the option

1. User invokes application: The user will invoke the code analyzer.

2. User provides the input on the console: The user will provide the input in the format File Path, File Pattern i.e. which patterns of the file the user wants to analyzer and user may provide various options like '/S', '/X' and '/R' to search for sub-directory, to find the relationships and to write the output in an XML format.

3. Parse user input and retrieve the files: The analyzer will scan the input on the console line and retrieve the files from the specified path. Only the files which match the pattern specified by the user are retrieved.

4. Analyze the types in file: The code analyzer would parse the first file. It will then find out the various types, the function sizes, etc.

5. Store the type information: The analyzer would then store the various information of the file in the repository.

6. Check if there are any more files left for parsing. If yes, parse the next file else proceed for the next step.

7. Find the relationships: Again parse the file and check for the pre-defined relationships.

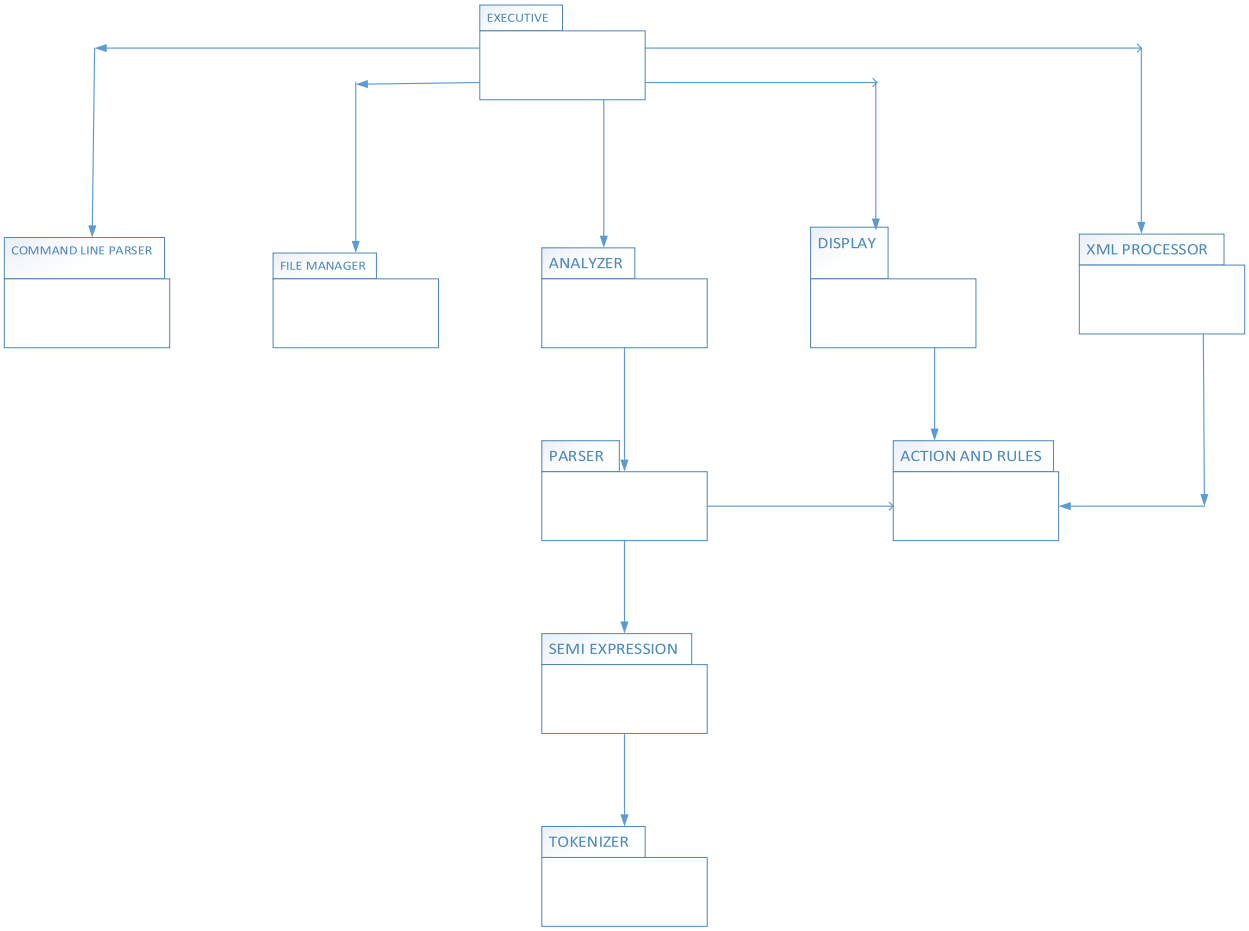
8. Check if there are any more files left for parsing. If yes, go for the next file else proceed for next step.

9. Retrieve the results and process the information.

10. Print the output to the console. If the user has provided an option for an XML file, also write the output to the file in an XML format.

6.PACKAGE/MODULE STRUCTURE

The following diagram represents the package structure of the code analyzer. There are total ten packages. Below is the description of the packages.



6.1 EXECUTIVE PACKAGE

Description: The executive package is the entry point for the code analyzer.

Responsibilities

- Shall co-ordinate with other packages in the analyzer.
- Shall be the first package which is called when the user starts the application
- Shall invoke methods from other packages. It will also provide outputs from one package as an input to other packages.
- Shall communicate with the command line parser package, file manager package, analyzer package, display package and the XML parser package.
- Thus, this is the driver package of the application.

6.2 COMMAND LINE PARSER PACKAGE

Description: The command line parser package is used to handle the input on the command line.

Responsibilities

- Shall parse the input provided by the user on the command line.
- Shall forward the path of the file set, patterns of the file to be used and the options provided by the user if any to the executive package, which will then forward it to the file manager.
- The user input should be of the type "File Path, File Pattern, Options"

6.3 FILE MANAGER PACKAGE

Description: The file manager package is used to get all the files from the path specified by the user.

Responsibilities

- Shall search for the files on the path specified by the user on the command line.
- Shall search only for the files which matches the pattern specified by the user.
- Shall perform a recursive search on the directory if the user has inputted the option of recursive search.
- Shall get the command line input from the executive package.
- Shall user the .NET framework file handling mechanism to carry out the tasks.

6.4 ANALYZER PACKAGE

Description: The analyzer package would co-ordinate with the parser and provide the files mentioned by the user to it.

Responsibilities

- Shall communicate with the parser package.
- Shall provide the files given by the executive package, provided to it by the file manager to the parser for parsing in a sequential order.

- Shall communicate with the parser twice, for parse one for finding the types and the scope and sizes of the function. Other for finding the relationships between the types.

6.5 PARSER PACKAGE

Description: The parser package would as the name suggests parse the files and execute a particular action based on a particular rule.

Responsibilities

- Shall get the required files to be passed from the analyzer.
- Shall communicate with the semi expression which in turn communicates with the tokenizer.
- Shall communicate with the rules and action package to look for the pre-defined rules and the associated action.
- Shall process the semi-expressions returned by the semi expression package based on the rules.
- Shall be used twice, once to find out the types, complexities, etc. and the other to find out the relationships between the types.

6.6 SEMI EXPRESSION PACKAGE

Description: The semi expression package generates as the name suggests semi-expressions i.e. sequence of tokens useful for code analysis.

Responsibilities

- Shall communicate with the tokenizer package to get the tokens generated by it.
- Shall get the files from the parser package and forward it to the tokenizer.
- Shall extract the semi expressions from the tokens and pass them to the parser package.
- Semi expressions are the right amount of code required to make a decision about a grammatical construct.

6.7 TOKENIZER PACKAGE

Description: The tokenizer package tokenizes the ASCII text files provided to it by the semi-expression package.

Responsibilities

- Shall construct tokens using rules which are appropriate for the code analysis.
- Shall get the file from the semi-expression package, generates tokens and gives it back to the semi package.
- Shall convert the file's words into punctuations and symbols.
- E.g. '\n', quoted strings, comments would be generated as tokens.

6.8 ACTION AND RULES PACKAGE

Description: The action and rules package contains the necessary rules required for code analysis along with the corresponding actions.

Responsibilities

- Shall receive a request from the parser. The parser package will communicate with this package and check the corresponding rules and the actions related with the rule.
- Shall be communicated by the parser in pass1 to check for the rules to identify the types i.e. class, structs, enums, and interfaces.
- Shall be communicated by the in pass2 to check for the rules to identify the relationships between various types.
- Shall use the ScopeStack to keep track of the scope of a particular type by pushing and popping the type, name and the place i.e. the line number.
- Shall contain rules to detect the types, complexities, relationships between types and the scope of the types.
- Shall communicate with the Repository class to pass data between actions and use the ScopeStack.
- Shall be communicated by the display package and the XML processor package to get the results.

6.9 DISPLAY PACKAGE

Description: This package is used to display the results on the console.

Responsibilities

- Shall display the result of the code analyzer to the console.
- Shall display error messages if any on the console.
- Shall be instantiated by the executive package.
- Shall get the results from the actions and rules package.
- Shall filter out the results if a particular file has no types or functions, relationships, etc.

6.10 XML PROCESSOR PACKAGE

Description: This package is used to write the output in an XM format

Responsibilities

- Shall write the output to a file in an XML format if asked by the user via the command line options
- Shall be instantiated by the executive package.
- Shall receive the results from the actions and rules package.

7.CRITICAL ISSUES

7.1 PERFORMANCE OF THE TOOL DUE TO LARGE SET OF FILES

A project contains large number of packages. Each package has many lines of code. This large set could slow down the analyzer. The performance could drop drastically if the user has included the sub-directory search as the tool will have to analyze the various types and the relationships between each and every types. Also as the application will have make two passes for each file, the time taken would be large.

Solution: One solution would be modify the analyzer code in such a way that only one pass would be required to find the types and their relationships. In the first pass whenever the parser finds the types, we could dynamically have another parser communicating with the repository and the rules and action package and finding the relationships between the types found. Thus, by checking the relationships dynamically we could reduce the time taken by the analyzer.

7.2 OUTPUT DISPLAYED TO THE USER

If the program has a lot of complexities and types, and if the output displayed is in a row format, the user might find it difficult to interpret. As the types and the complexities goes on increasing, the number of rows will go on increasing and it might not be a good picture for the user. Mapping the output in a format which is useful for the user is very important.

Solution: One solution would be to sort the output before displaying to the user. We can use a graphical representation to display the various relationship. Also, we could sort the output in an ascending namespace order and then display the various types and relationships under single namespace in a graphical manner. If there are types which are cross referenced in various namespaces, those could be shown separately. Also, we could separately highlight functions which have a very large scope and scopes which involve lot of complexities. Thus by properly dividing the output in a logical manner, we could make it useful for the users.

7.3 LOGICAL/SYNTACTICAL ERROR IN THE USER PROGRAM

Suppose there is a logical or syntactical error in the user code i.e. a scope is opened and is not closed, or the scope elements are used outside a particular scope. In this scenario, our analyzer won't be able to correctly analyze the code and it might result in improper analysis.

Solution: A simple solution to the above problem would be, our analyzer should continue with the analysis and not break due to the syntactical error. It should give the output for rest of the types/relationships but it should also tell the user that a particular scope has

encountered an error and it should tell the user to re-check that particular scope/construct.

7.4 FILE IS PROTECTED BY ACCESS RIGHTS.

There could be a scenario where the files present on the path provided by the user are protected and the analyzer is not able to open and retrieve the information.

Solution: A simple solution to the above problem would be, prompt the user about the issue in the very beginning when the file manager throws this error, so that the user can change the access rights and then give the file for analysis.

8.SUMMARY

Code Analyzer would be a very helpful tool both in small size and large size projects. It could drastically reduce the time required to understand a project and thus would be helpful in numerous ways like for knowledge transfer, for enhancements, for making small changes in the code, etc. We have discussed the overall application flow along with the user set of the application. We have also discussed few critical issues and how they could be handled. We have discussed the technical package diagram and the various interfaces between packages. Thus, to sum-up, code analyzer is a very useful tool for developers and it will make a developer's life easy.

9.APPENDIX

SAMPLE OUTPUT OF THE PARSER.

```
 Type and Function Analysis
 --------------------------

 line# 44   entering   namespace CodeAnalysis
 line# 49   entering     class Parser
 line# 53   entering       function Parser
 line# 56   leaving        function Parser
 line# 57   entering       function add
 line# 60   leaving        function add
 line# 61   entering       function parse
 line# 72   leaving        function parse
 line# 73   leaving      class Parser
 line# 75   entering     class TestParser
 line# 79   entering       function ProcessCommandline
 line# 95   leaving        function ProcessCommandline
 line# 97   entering       function ShowCommandLine
 line# 106  leaving        function ShowCommandLine
 line# 112  entering       function Main
 line# 159  leaving        function Main
 line# 161  leaving      class TestParser
 line# 162  leaving    namespace CodeAnalysis

 locations table contains:
  namespace,              CodeAnalysis,    44,   162
      class,                    Parser,    49,    73
   function,                    Parser,    53,    56
   function,                       add,    57,    60
   function,                     parse,    61,    72
      class,                 TestParser,    75,   161
   function,        ProcessCommandline,    79,    95
   function,           ShowCommandLine,    97,   106
   function,                      Main,   112,   159
```

The above screen shows the sample output of the parser. The parser has been provided to us by Dr. Fawcett. The parser has been partially implemented and analyses the class, namespace, functions and scope.

10.REFERENCES

1) Tokenizer handout, provided by Dr. Fawcett
2) Cover Sheet-Parser, Semi-expression, provided by Dr. Fawcett
3) GaCheng OCD, provided by Dr. Fawcett