

# Code Analysis OCD

Project #1

9/17/2014  
David Lambrych  
dalambry@syr.edu

**CSE 681- Software Modeling and Analysis**

# TABLE OF CONTENTS

|                                                |   |
|------------------------------------------------|---|
| Executive Summary.....                         | 3 |
| Introduction.....                              | 3 |
| Uses .....                                     | 3 |
| Users (Actors) .....                           | 4 |
| Software Architect (SWA) .....                 | 4 |
| Responsible Individual (RI).....               | 4 |
| Quality Assurance Engineer .....               | 4 |
| Individual Contributor (Developer).....        | 4 |
| User Interface .....                           | 4 |
| Structure .....                                | 5 |
| Package Diagram.....                           | 5 |
| Executive .....                                | 6 |
| CLParser .....                                 | 6 |
| File Manager.....                              | 6 |
| Navigator .....                                | 6 |
| Analyzer.....                                  | 6 |
| Parser.....                                    | 6 |
| Semi .....                                     | 6 |
| Toker .....                                    | 6 |
| Repository.....                                | 6 |
| ActionsAndRules .....                          | 6 |
| Display.....                                   | 7 |
| XML Writer .....                               | 7 |
| Code Analyzer High Level Activity Diagram..... | 8 |
| Description .....                              | 9 |
| Validate Command Line .....                    | 9 |
| Build File List.....                           | 9 |
| Navigate Sub Directories .....                 | 9 |
| Parse Files .....                              | 9 |
| Build Type Relationships.....                  | 9 |
| Display Results.....                           | 9 |

Convert to XML and write to file..... 9

Critical Issues ..... 10

    Analysis of Very Large Baselines..... 10

    Avoiding Parallel Processing issues..... 10

    Error Handling..... 10

## EXECUTIVE SUMMARY

This document presents the design and architecture of the Code Analyzer program. Given a set of C# files, the application will provide function and “type” information, where “type” represents a C# namespace, class, interface, or struct. The program will allow the user to assess the files’ structure, size, complexity, and “type” relationships.

The Code Analyzer is decomposed to a set of specialized packages, where each package represents one or more object oriented classes. The purpose of each package is to represent a logical grouping of related functionality, or tasks, to accomplish the objectives of the Code Analyzer program. This includes traversing file tiers, parsing files, establishing rules and actions for function and/or “types” found, and displaying/storing the analysis.

Code Analyzer’s intended users are Software Architects (SWA), Quality Assurance Engineers (QA), Software Development Responsible Individuals (RIs), and Individual Contributors (IC) to the software’s development. The Code Analyzer will aid the users when writing new code to ensure the code meets criteria in Software Standards and Procedures Manual (SS&PM), performing Engineering Technical Reviews (ETR), or when coming up-to-speed with unfamiliar code baselines.

Design considerations and solutions will be given for Code Analyzer's performance and performance limitations. This includes the length of time to perform analysis, what will happen when C# files containing syntax errors are encountered, and incorporating parallel processing where applicable.

## INTRODUCTION

The Code Analyzer will comprise functionality to:

1. Evaluate command line data passed into the program.
2. Manage directory hierarchies and C# files within the hierarchies.
3. Perform analysis of C# files utilizing command directives passed into the program.
4. Data storage pertaining to type names, functions pertaining to the types, the complexity of these functions, and the correlation of type relationships across file boundaries.
5. Provide analysis results to the user and, when commanded, save the analysis results to an XML file.

The Code Analyzer architecture will be decomposed into packages, each containing a related set of responsibilities. The architecture will contain the high level packages Executive, Command Line Parser, File Manager, Analysis, ActionsAndRules, Repository, and Display. In addition, a set of supporting packages will be defined to assist in the implementation of the high level packages’ behavior. The rational and details of the packages will be fully described in a following sections of this document.

## USES

The Code Analyzer will be used to evaluate the cyclomatic complexity and function size of C# code being developed for projects of various sizes. Several key participants of the development team will utilize the program such as the SWA, RI, QA, and IC.

### USERS (ACTORS)

#### *SOFTWARE ARCHITECT (SWA)*

The SWA will utilize the tool to gather data for reports, proposals, and provide comments for ETRs. Given the number of developers and amount of review materials on large scale programs, the SWA will utilize the Code Analyzer to ensure implementation is adhering to the design and within the complexity criteria outlined by the SS&PM.

#### *RESPONSIBLE INDIVIDUAL (RI)*

The RI manages a development package and the developers working on the package. The RI often provides complexity and design information to the Software Program Engineer (SPE) for Monthly Software Engineering Program (MSEP) reviews. The Code Analyzer program will help the RI ensure the quality and consistency of the package is as expected. The RI must efficiently assess the code for the package to verify the development is within defined cyclomatic complexity and function size specified documents governing the development such as the SS&PM.

#### *QUALITY ASSURANCE ENGINEER*

The QA Engineer will utilize the Code Analyzer to ensure the integrity of the developed code. The engineer will need to perform analysis on several packages and/or baselines throughout the development cycle. The Code Analyzer program will allow the engineer to perform review tasks efficiently by generating reports of specified baselines' size and cyclomatic complexity.

#### *INDIVIDUAL CONTRIBUTOR (DEVELOPER)*

The Code Analyzer is central to the IC. The developer will utilize the program to make sure newly developed code is within size and complexity constraints. The Code Analyzer will be used to provide size and complexity information as part of the ETR review materials. The IC should also re-run the analyzer after incorporating comments generated from reviews to ensure the modified code has not exceeded specified boundaries. In addition, the Code Analyzer program can be used to help the IC come up to speed on unfamiliar packages that may be reassigned.

### USER INTERFACE

The Code Analyzer program will be invoked via the command line. This is the simplest format for running the program and also allows the program to be easily incorporated into a development suite which could include tools to evaluate memory leaks, race conditions, error handling issues, as well as the Code Analyzer's functionality.

On the command line, the Code Analyzer program will be invoked as follows:

```
% codeAnalyzer <baseline path > </S> </R> </X>
```

Where,

Default (no options specified on command line) – C# files in the current directory will be evaluated and type names, their function(s) name, function size, function complexity, and a summary of type and function data is displayed to the console.

<baseline path> – the directory for the file(s) under analysis

</S> – all subdirectories of the baseline path are to be evaluated.

</R> - display all type relationships in the files set under analysis instead of the function size and complexity.

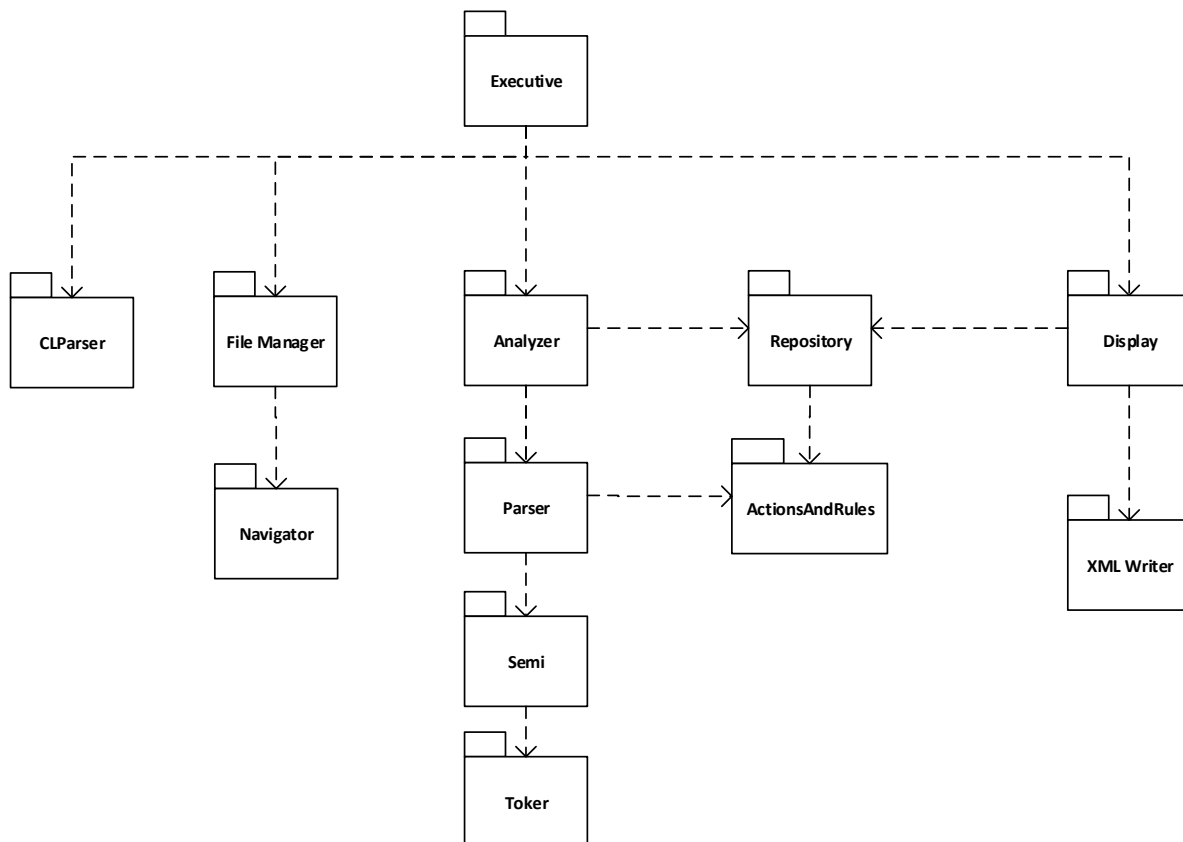
</X> - save analysis data in a file using XML formatting in addition to displaying data to the console.

Note: Although the Code Analyzer can be run from the command line, a more efficient way to run the program is using a batch file as outlined in the specifications of Project #2.

## STRUCTURE

### PACKAGE DIAGRAM

The Code Analyzer package diagram depicts the modules and their interaction for the functionality comprising the program. The following sections describe each package.



### *EXECUTIVE*

The Executive package contains the “main” and will receive commands issued by the user via the command line. All other modules are created and invoked via the Executive, either directly or indirectly.

### *CLPARSER*

The CLParser package is responsible for deconstructing the command line input, assessing the input string for valid command line options, and returning directory path/file string(s) to the Executive.

### *FILE MANAGER*

The File Manager package is responsible for traversing a directory tree and iterating through files of a folder. It builds a list of files for analysis by the Analyzer package. The Executive provides the directory path/file string(s) to the File Manager.

### *NAVIGATOR*

The Navigator package handles the details for traversing sub directory paths when the </S> option is specified on the command line.

### *ANALYZER*

The Analyzer package is responsible for examining a list of files provided by the Execution package. The package utilizes the Parser package to realize this behavior. Additionally, Analyzer evaluates type relationships across the file set and provides the information to the Repository package.

### *PARSER*

The Parser package is a container of rules, where each rule is a grammatical construct. Each rule has an associated set of actions pertaining to its construct. Parser utilizes the services of the ActionAndRules, Tokener and Semi modules to realize this behavior.

### *SEMI*

The Semi package provides facilities to extract semi expressions. A semi expression is a sequence of tokens that is just the right amount of information to parse for code analysis. Semi expressions are token sequences that end in "{" or "}" or ";"

### *TOKER*

The Tokener package provides the facilities to tokenize ASCII text files. That is, it composes the file's stream of characters into words and punctuation symbols.

### *REPOSITORY*

The Repository package provides a container for data pertaining to a parsed file and for type relationships. The Repository utilizes the services of the Analyzer, Parser and ActionAndRules packages to determine the relevant datum for the Code Analyzer program.

### *ACTIONSANDRULES*

The ActionAndRules package contains a defined set of behaviors for constructs utilized in the Code Analyzer program. ActionsAndRules utilizes the Display module to present the statistical results to the user.

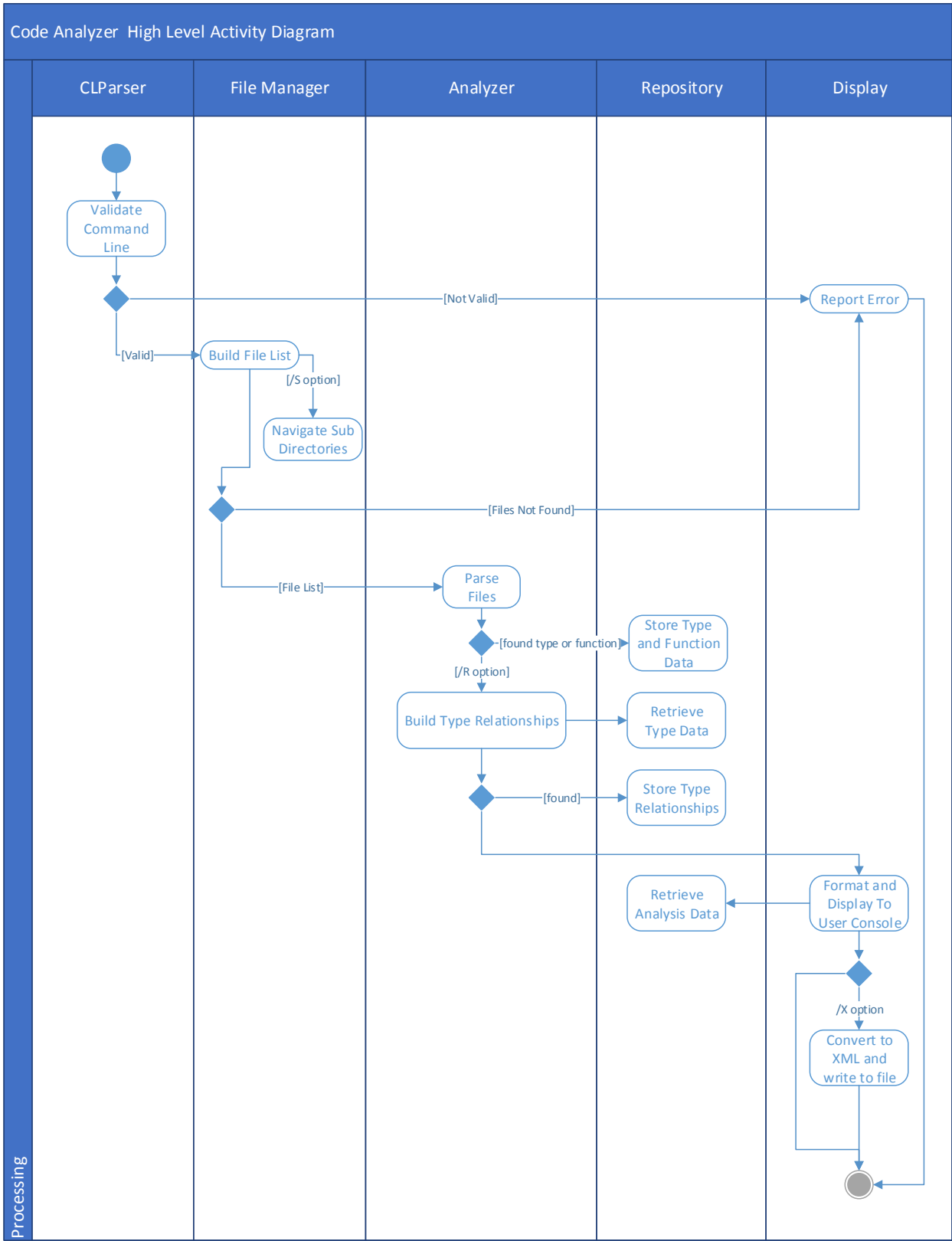
### *DISPLAY*

The Display package provides the facilities to display formatted information to the user's console.

### *XML WRITER*

The XML Writer package provides the facilities to save the display data to an XML formatted file.

# CODE ANALYZER HIGH LEVEL ACTIVITY DIAGRAM



### DESCRIPTION

The diagram shows the high level activities that comprise the Code Analyzer program. Upon invocation of the program, the command line arguments are parsed and validated. If the command line does not contain a valid set of options error text is displayed to the user console followed by text describing the valid command line arguments that are accepted by the program. After validation, a list of files is compiled. When the “/S” option is specified on the command line, sub directories are traversed and files from the subdirectories are added to the file list. If there is not at least one C# “.cs” file in the directory, or subdirectors when the “/S” option is specified, an error is displayed to the user console. An analysis is then preformed on the file list by parsing each file for type and function data, including the cyclomatic complexity and size for each function found. This data is then stored. If the “/R” option has been specified on the command line, an analysis is then performed to determine the type relationships. When all files in the file list have been parsed and analyzed, the results are displayed to the user console. If the “/X” option has been specified, the formatted data is written to a file in XML format.

### *VALIDATE COMMAND LINE*

When the Code Analyzer receives command line input from the user, the program must ensure the arguments are useable. This activity checks the input against a defined set of rules. If the commands are not valid, an error condition is raised.

### *BUILD FILE LIST*

Add files with the “.cs” extension to a file list.

### *NAVIGATE SUB DIRECTORIES*

Find all subdirectories in the current working path, traverse the given subdirectory, and add files with the “.cs” extension to the file list.

### *PARSE FILES*

Look for a predefined grammatical constructs in a file. When found, perform an action based on a predefined set of rules and save the result.

### *BUILD TYPE RELATIONSHIPS*

Using the parsed data, determine if the

### *DISPLAY RESULTS*

Retrieves type, function, and type relationship data, formats the data, and sends the formatted data to the user console.

### *CONVERT TO XML AND WRITE TO FILE*

The type, function, and type relationship data is formatted to XML tags and written to a file.

## **CRITICAL ISSUES**

### **ANALYSIS OF VERY LARGE BASELINES**

Some development projects have extremely large code baselines. This may be a problem for users needing analysis results in a short timeframe. The development of the Code Analyzer program should be focused on parallel processing where it makes sense, such as when retrieving files and parsing them. Additionally, an option can be provided, i.e. through an option flag at the command line, which performs a quick analysis of the tiers and provide a report to the user for the length of time it will take for a full baseline analysis. This allows the user to plan for the allotted time.

### **AVOIDING PARALLEL PROCESSING ISSUES**

The developer needs to pay special attention to the parallel processing indicated for baseline comparison. Care needs to be taken not to introduce race conditions, memory issues, or thread priority issues. For example, when utilizing the Repository, thread safe code should be utilized to protect critical regions from corruption during the update of member variables.

### **ERROR HANDLING**

The Code Analyzer program needs to account for error conditions. This includes syntax errors within files, corrupted files, permission issues for directories within tiers being traversed, and group or file permissions. The program needs to be robust enough to recover from such errors and generate a report for suspect files or directories. Encountering errors should not prohibit the analysis of valid files or directories. The Repository should provide facilities to store error conditions and saved data should be utilized.