CSE681 OCD

Remote Package Dependency Analysis

CSE681

XIAO TAN

Syracuse University

December 5,2018

Contents

1. Executive Summary

Remote Package Dependency Analysis is a Client-to-Server system that we can send message to server and get the feed back on GUI, to do a Dependency analysis and Strong Component detection on remote files, so we can easily analyze large project codes

The Remote Package Dependency Analysis contains 9 Major parts:

Tokenizer: extract the tokens from the codes, separate the code lines;

SemiExpression: collect the tokens from tokenizer and combine them into semiExpresions, this part help users do a syntax analysis on the code, and put those information into different sorts, to make our later analysis easier.

DependencyTable: use the semiExpressions from the last step to find the internal relationships between the target code files, it can find the relationships between classes and packages, so that we can draw class diagram and package diagram.

StrongComponent, find the Strong-Component relation in those files.

Client: as a GUI for users to send their request to the Server, and get the feedback on GUI, Which means it is very concrete.

Server: work as a server to handle the request from client, different request consistent in different operations, after the operation, the server send a reply to the Client to add the operation information.

Comm help build a communication system bring about the message transportation, it define the data structure and the rules a message should be, both Client and Server have a Comm part to communicate with each other. Comm part contain Sender and Receiver.

Test part help developers to evaluate the stability and accuracy of this project.

FileMgr help developers to select files and do any file operations as the wish.

2. Introduction

Finishing a design of a big software system is a tremendous workload, there are tens of thousands of code lines that need us to write and tests, no matter how brilliant a man can be, he will not be able to remember all the details, and large software system are build through collaboration, all designers and tests need to know the information to get involved in that project. How can we make sure designers and tests can find the information they need efficiently?

When late new parts added to software system baseline, we have to find and fix

the potential problems, we will re-run test sequences for those parts and, perhaps, for the entire baseline. But do we really have to waste time on unrelative code test?

To solve the problems above, we need to partition code to into relatively small part and completely test each of them before inserting them to baseline, and we need a powerful tool to help us to manage such a big software project, a process that resembles code analysis.

Briefly speaking, we need two main steps to deal with the code: token the code and extract semi-extraction. So we can get all the important information we want and those information are simplified enough to be concise. The results of lexical scanner are the key code lines that big software system developers need to pay more attention.

After we get the SemiExpressions and tokens, we can do the dependency and Strong Component detection, then we can know the internal relationships, it can make us better analyze the large software System.

And this time we need to build a Client-To-Server System to do a remote analysis on the project code, we need to do a dependency analysis and strong component detection on the project code, the main process is to send request message to the server through our Client GUI, after the Server take the message it can call the child process to execute and put the result through WCF back to the Client, so we can directly get the feedback.

3. Uses

As I have mentioned above, this Scanner can serve different users: developers, test, customers and managers. Each kind of user have different need for this Scanner.

3.1 Developers

Developers are the users who design this Client-to-Server dependency Analysis, during the development, developers need to get the feedback from the two key functions(Tokenizer and SemiExpression) respectively so that they can directly measure the results, when developers use Client to call those two packages, the two packages would return their results back to the Client GUI. It is convenient for the developers to know the progress status and the usability of this project.

Tokenizer and SemiExpression should leave two interface for the ClientEXEC in order to better serve developers.

3.2 Customers

Customers are the normal users of this System, so we need to assume that they don’t need to know the structure and principles of this tool, all they can do is just run this tool on their big software system and get the results to better develop or

implement their system. According to the sequence the customers use Client GUI to send the request to the server, So Server call SemiExpression to find the dependencies and strong component, and SemiExpression need to call Tokenizer to get the tokens, Tokenizer need to call CodeResources to get code lines.

The procedure of Customers is one single line, usually the customers don't have the right to make changes to this Lexical Scanners.

3.3 Test

The Test part is to measure the result of the remote analysis system , Communication part's stability and usability. During the development or after the later implement, the test part can respectively test each function, if errors occur we can quickly find and fix them, it is beneficial to the robust of this Scanner.

3.4 Managers

Some user may not be satisfied with the functions that the customer-orientated scanner provides. For instance, if we want to change the state inside the tokenizer, change the rules of the grammar,make this scanner more extendable, we may need privilege to edit the rules. For those people we call them managers, managers have more right than customers, they can redefine the rules, make this scanner not only suit one specific programming language.

4. Packages

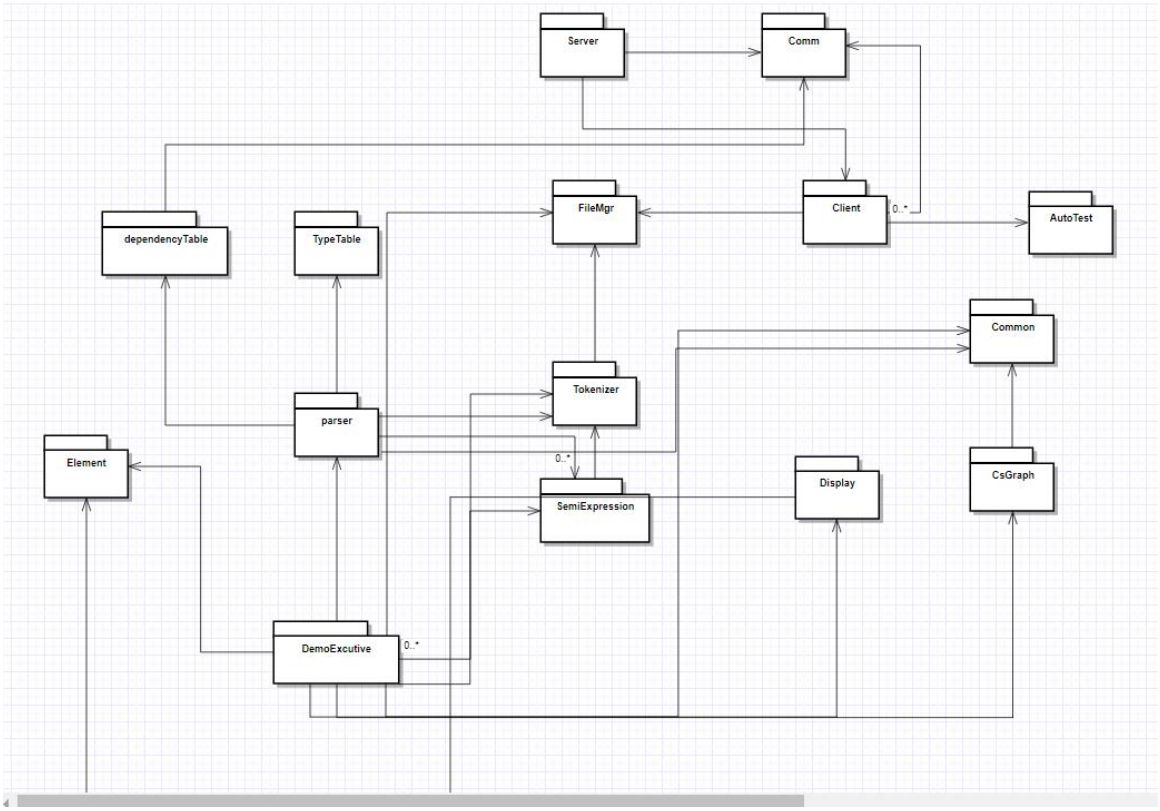


Figure 1:Package Diagram

For this System I divide it into fifteen parts: Element, DemoExecutive,Parser, TypeTable, dependencyTable, SemiExpression, Tokenizer, FileMgr, Server, Comm, Client, AutoTest, Common, CsGraphs, Display.

4.1 Element

This module defines the Elem class, which holds:
- type: class, struct, enum
- name
- code location: start and end line numbers
- size and complexity metrics: lines of code and scope count

4.2 DemoExecutive

DemoExecutive finds files to analyze, builds typetable, by parsing files for defined types. builds dependency table by parsing files. And using typetable file and namespace information builds dependency graph from dependency table and analyze

strong components.

4.3 Parser

Parser package contain a lot of rules and actions, when it use semiExpression to get the semis, use it to compare with the rules, each rule has a list of actions, semi will be lead to actions, After operate those semis, the result will be stored in the TypeTable.

4.4 TypeTable

TypeTable package manages type information generated during type-based code analysis. Store the types and return the information related to the types.

4.5 dependencyTable

After TypeTable collect all the types, the parser will traverse the types and find the dependencies between different classes and packages.

4.6 SemiExpression

This package contains a Semi class that implements ITokenCollection interface. It also contains a Factory class that creates instances of Semi. Semiexpressions are collections of tokens that are useful for detecting specific grammatical constructs. It is important that each instance of Semi contains all tokens necessary to make a decision about a grammatical construct, It is also important that each Semi instance does not contain tokens relevant for more than one detection.

4.7 Tokenizer

This tokenizer is implemented with the State Pattern, and Collects words, called tokens, from a stream. Discards all whitespace except for newlines which are returned as single character tokens. By default, collects and discards all comments, but has an option to return each comment as a single token. Also returns quoted strings and quoted characters as tokens. Toker correctly handles the C# string @"...". This package demonstrates how to build a tokenizer based on the State Pattern. Class Diagram

4.8 FileMgr

Recursively displays the contents of a directory tree rooted at a specified path,

with specified file pattern. This version uses delegates to avoid embedding application details in the Navigator.

4.9 Server

This package defines a single NavigatorServer class that returns file and directory information about its rootDirectory subtree. It uses a message dispatcher that handles processing of all incoming and outgoing messages.

It also contains a SpawnProc class that is used to call a child process, specifically in this project, we need to call a Dependency analysis and strong component detection process to get the result.

4.10 Comm

This part defines the data structure of the message that need to be transported, and define a sender and receiver part to send and receive message, this part also define the Server and Client's environment and a blocking queue to make the message sending and receiving in order.

4.11 Client

This package defines WPF application processing by the client. The client displays a local FileFolder view, and a remote FileFolder view. It supports navigating into subdirectories, both locally and in the remote Server. In my solution, the Client GUI can also select specified files to the server to analyze, which make the whole process more flexible.

4.12 AutoTest

AutoTest is a package to run the sending messages and receiving messages automatically, so this could be used to meet some specific requirements.

4.13 CsGraphs

Csgraph package is a normal graph data structure, in this project it also has a StrongComponent function to detect the strong component relationships in the graph, those node with a directed edge which can construct a cycle, That cycle can be regarded as a strong component.

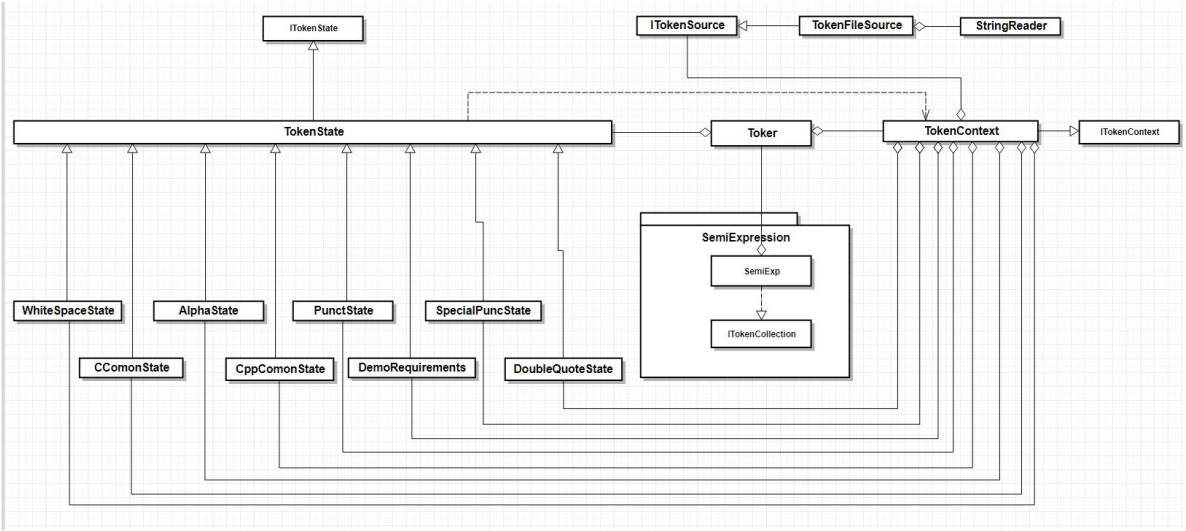
4.14 Display

Display manages static public properties used to control what is displayed and

provides static helper functions to send information to MainWindow and Console.

5. Classes

5.1 SemiExpressions and Tokenizer



Semi and Toker Class Diagram

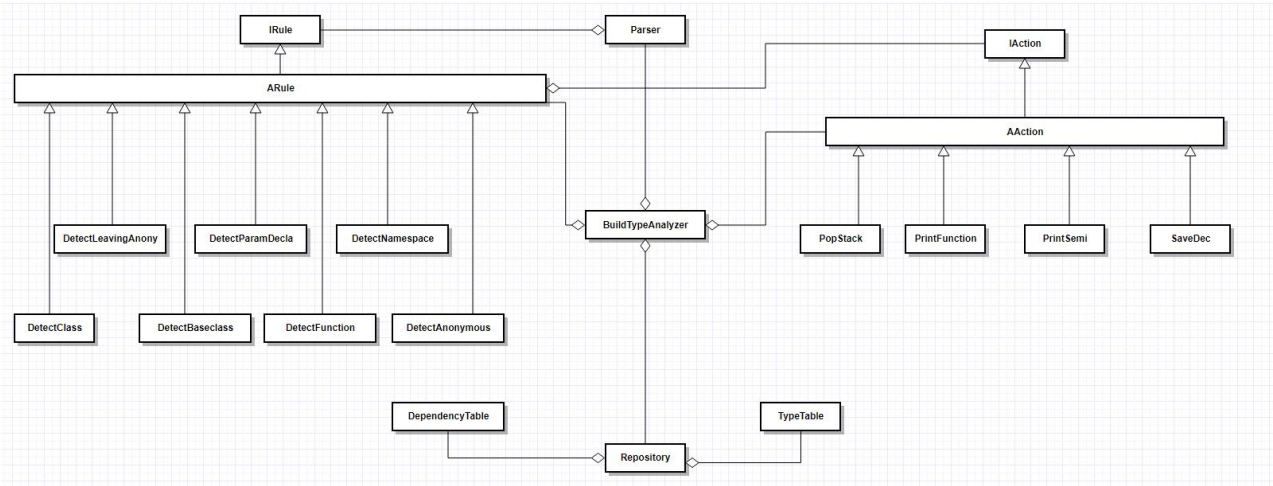
I built the Class Diagram, in this diagram I showed the classes of Tokenizer and SemiExpression.

The Tokenizer has three core classes and three interfaces: Classes include Tokenstate, Toker and TokenContext. Interfaces Include ITokenState, ITokenSource and ITokenContext.

TokenContext get the filesources, Tokenstate works like template in C++ to certify current status, Toker are the execution class.

SemiExpression call Toker.getTok() to get tokens and use ITokenCollection interface to collect Tokens, while meeting terminating characters the grouped tokens will be created as SemiExpression.

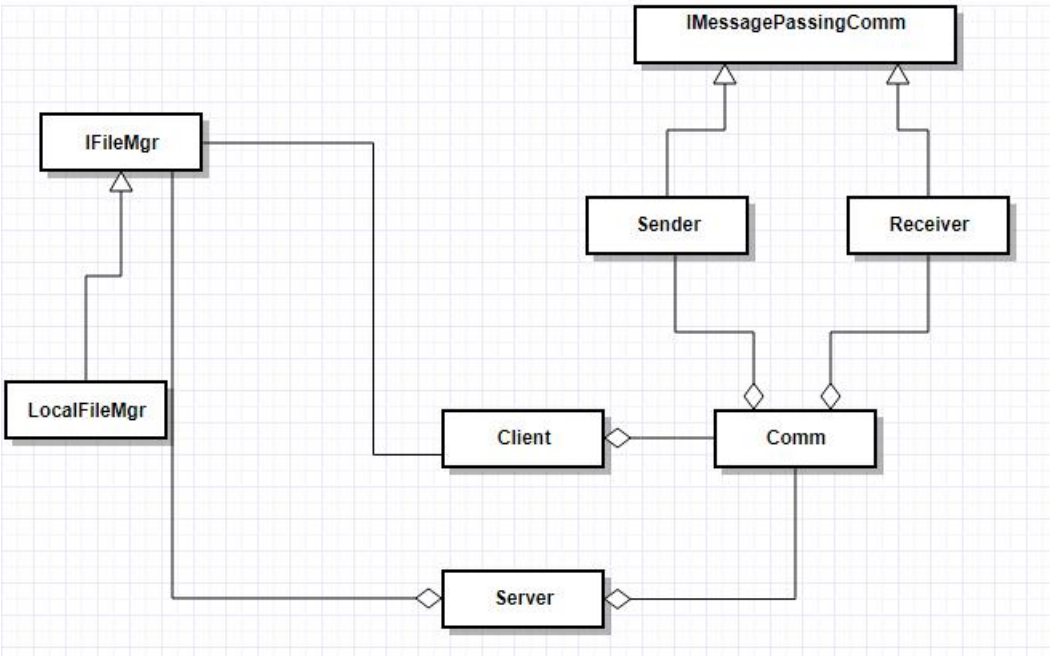
5.2 Dependency and StrongComponent



Dependency

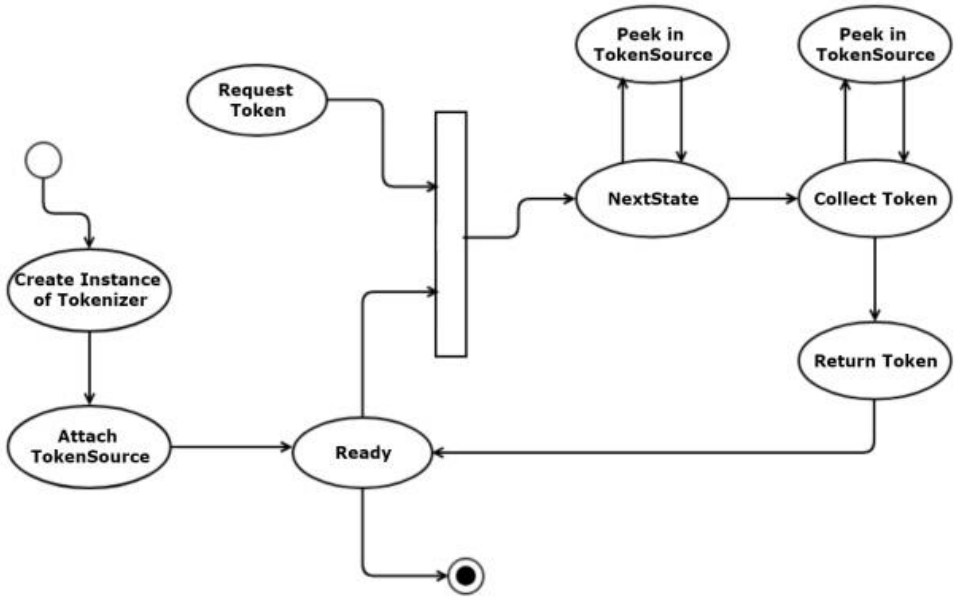
The Parser package include rules and actions, they are both inheriting their interfaces IRule and IAction, each rule has a list of actions, when the semi Expression is in the rule, it will face the consistent actions. While the BuildTypeAnalyzer class can build a parser class which can finally generate Dependencies and Types.

5.3 Client and Server

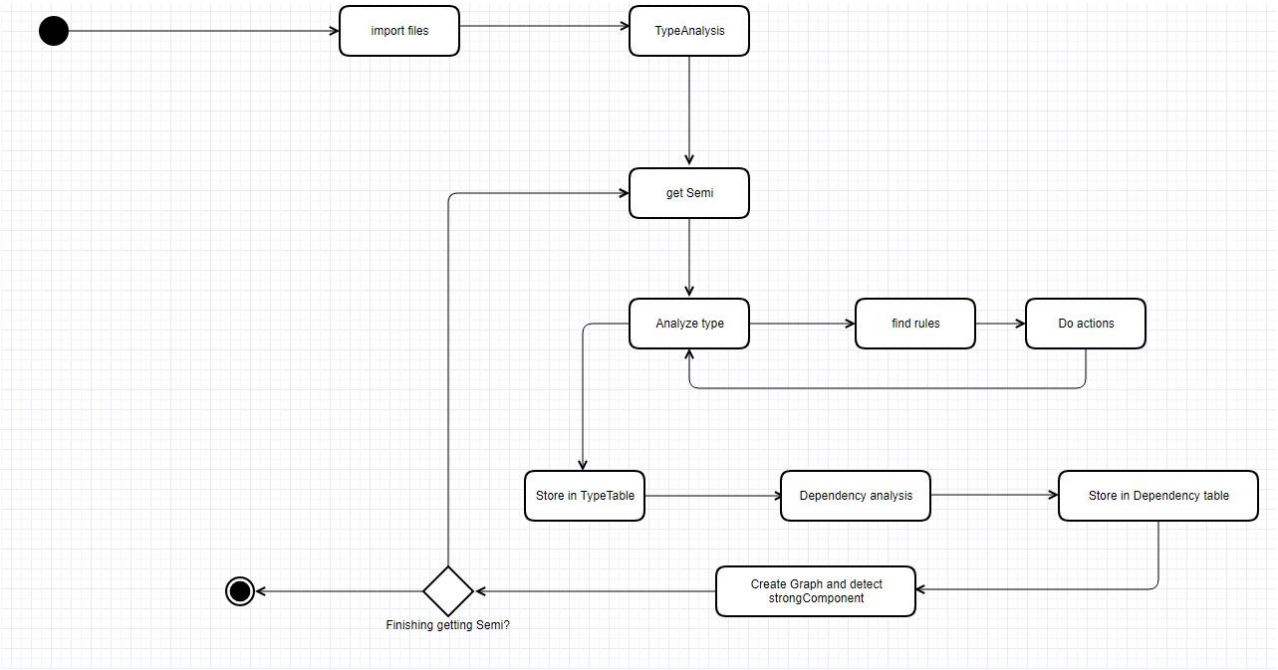


6.Activities

6.1 Toker-Semi

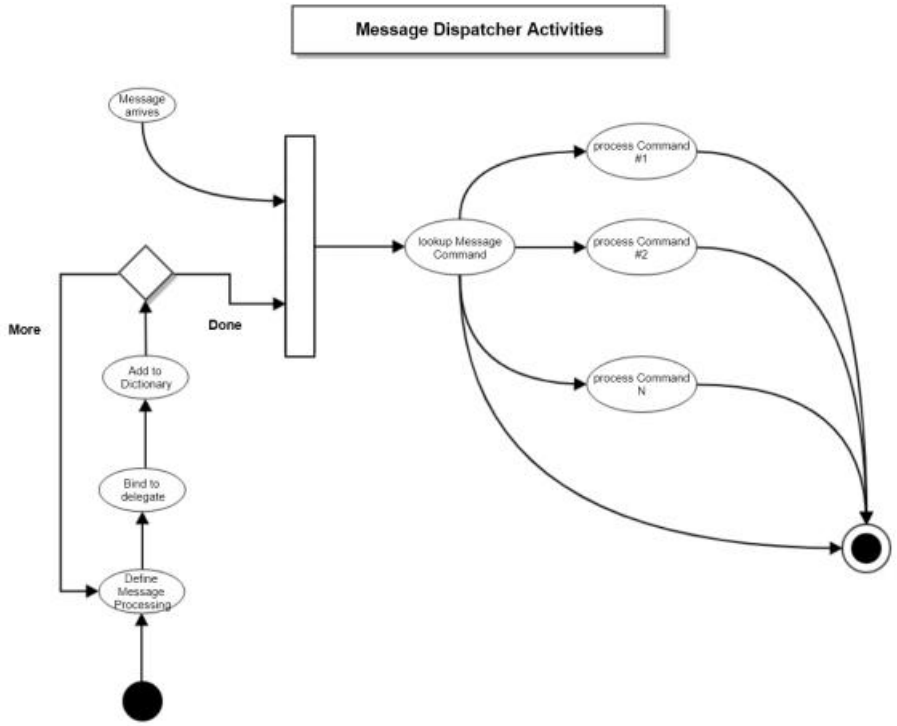


6.2 Dependency-StrongComponent



6.3 Client-Server

A message dispatcher can be implemented with a Dictionary<Msg.command, Action<Msg>>[3]. The Msg.command defines the type of processing needed for a message, and the Action<Msg> defines the processing used to handle that message.



7 Critical Issues

7.1 Errors

It is common to see the running errors while starting this scanner, the compiler may send messages to us but they are usually not complete. Some errors may happen due to the imperfect structure, some may happen due to the inconsistent version. How do we locate the error easily?

I suppose the test part to include such a function, it can not only test the stability and usability but also can detect the potential risks of this scanner. This correction make the scanner robust.

7.2 User Friendliness

For this Lexical Scanner, I assumed four kinds of users, how do we divide them

into four parts? How do we make sure all the users can get what they want while guaranteeing the security?

I suppose to make safety precaution on ClientEXEC to guide the users, the developers and managers need their id and password to identify themselves, and test part's execution program are not on ClientEXEC, it is in the testing program.

7.3 Usability and Stability

How do we make sure this program works well on the codes? To build a perfect Tokenizer is not an easy thing, even the greatest diamond has its flaw, this scanner may get the wrong result after this tool is released to customers.

Of course we can't avoid all the errors, but we can keep updating, all the users can feedback their emergencies back to us, we may fix the problems and release the newest version as most companies did.

8.Reference

http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/OCD_Samples/AkshithaKoganti.pdf

http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project1-F2018.htm

http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/StudyGuideOCD.htm