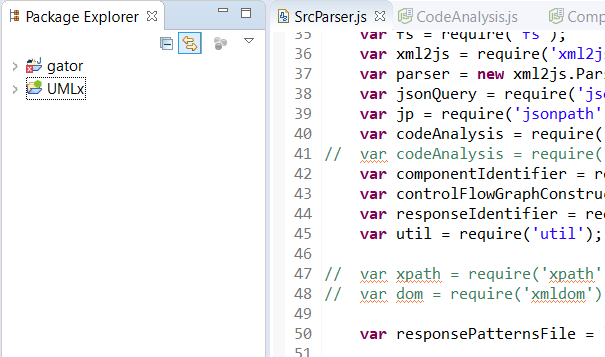
1. Install Eclipse from here:

<https://drive.google.com/open?id=0Bx6SYedbAp1CWmtmcXhXemI4SzA>

1. Import UMLx and Gator project (UMLx\facility-tools\GATOR\_Tool\gator-3.5\gator) into eclipse:



1. Compile gator:

export GatorRoot=/mnt/h/ResearchSpace/ResearchProjects/UMLx/facility-tools/GATOR\_Tool/gator-3.5

export ANDROID\_SDK=/mnt/h/ResearchSpace/Android\_Workspace/Android\_SDK (download…)

cd $GatorRoot/gator

./gator b

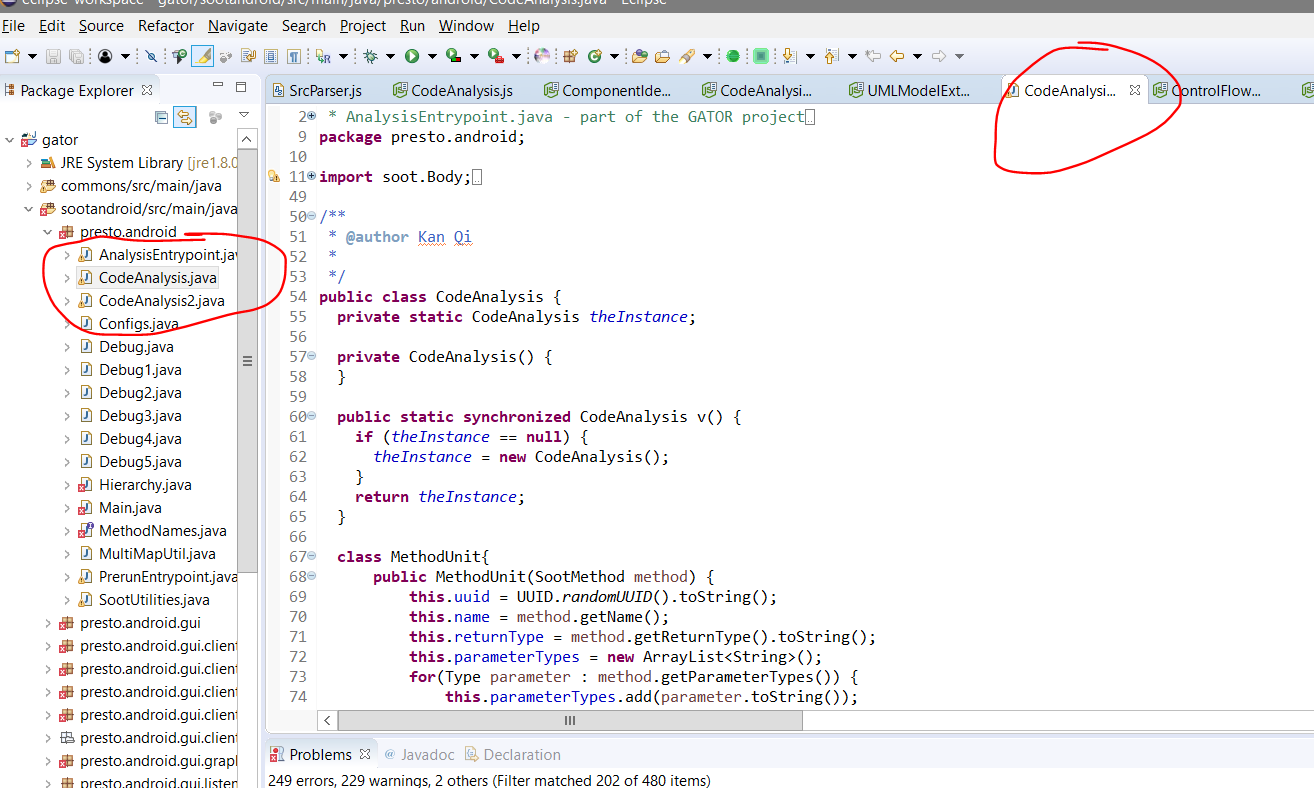
* If you are using windows, please follow the document to install ubuntu console following the document:

UMLx\facility-tools\GATOR\_Tool\gator-3.5\docs

1. Runt the command to see if your changes are within the console:

./gator a -p /mnt/h/ResearchSpace/Repositories/Android\ Projects/ModelAnalysisTest/app/build/intermediates/instant-run-apk/debug/app-debug.apk -client GUIHierarchyPrinterClient

The changes should be made to the following file:



1. Run the following command to see the analysis data:

node ./UMLxAnalyticToolKit.js "./data/OpenSource/kdm.xml" "./data/OpenSource/debug" "kess"

The data is generated in “./data/OpenSource/debug”

1. Compare the classification results with the existing ones.

Compare the classification results with the ones generated by Arcade.

1. See the change with respect to the performance in predicting project effort.

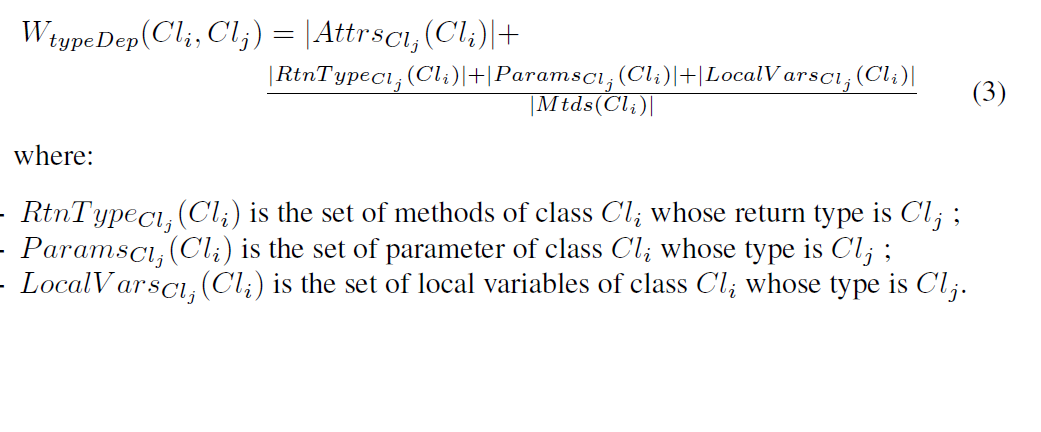
Source Code Analysis

Source code analysis (bytecode analysis) is to analysis the program level elements including classes, methods, attributes, calls, and dependencies. The first missions are to construct the following graphs from Android APK files using Soot. Soot is a program analysis tools which parse the information from source code, in our case, APKs. Remember that you don’t really need to fully understand Soot, you only need to understand its concepts and use some of the functions to achieve the purpose. The soot has already setup in the Gator project, what we do is just to understand what information is available at the CodeAnalysis.java file to develop the functions. Here you can find the APIs of soot: [*https://www.sable.mcgill.ca/soot/doc/index.html*](https://www.sable.mcgill.ca/soot/doc/index.html)

Here I briefly introduce the definitions of the graphs that we need to construct. The formal definitions are provided in this paper “Component-based Architecture Recovery from Object

Oriented Systems via Relational Concept Analysis”

***TypeDependencyGraph:*** A type of dependency between java classes. The formal definition is in below. Basically, we need to evaluate a kind of relationship between two classes. There are three situations, 1 one class C1 has a method M1 that has a return value typed as another class C2, 2 similarly to 1, the reliance is based on the type of parameter types of a method. 3. C1 may have a variable typed as class C2. Identify the relationships using soot and create edges between classes (as nodes) based on those to construct the graph.



***ExtendsGraph:*** Similar to the type dependency graph, it shows the relation between two classes by extension. For example, if A is a super class of B, then B extends A (B -> A). Interface implementation should be excluded.

***CompositionGraph:*** Composition is defined as class A is used as an attribute of class B. So if for any two classes, there is such relationship, create an edge.

For those graphs, there are counterparts in the following file. You may get some ideas of how to construct the graphs.

