MySQL Dependency Extraction

Ante Pimentel (212166187)

Dusan Birtasevic (210451383)

Francis Okoyo (213811336)

Hashim Al-Helli (212172359)

Richard Van (212918652)

York University

Table of Contents

Abstract

This Report illustrates different methods of dependency extraction. Three different dependency extraction methods are compared and contrasted; Understand, Include, and srcML. The methods are compared quantitatively and qualitatively. The quantitative comparison covers details such as: total number of dependencies, dependencies extracted by each method, and exclusive dependencies extracted by each method. The qualitative comparison covers the usefulness and completeness of the Include and srcML methods via Precision and Recall and F-Measure. Limitations, implications, and risks are covered and mentioned following the comparisons.

1.0 Introduction and Overview

When attempting to recover the architecture of a large scale software system it is generally impossible to go through each source code file manually and try to deduce the components of the system and its associated connections. One approach is to extract the file dependencies and use a containment file to generate a visual representation of a system's architecture. Assignment 2 focused on creating a containment file from already extracted dependencies to visualize the architecture of MySQL. Here, we will be covering how the source code dependencies are extracted.

The purpose of this report is to give the reader a better understanding of how dependency extraction works, the different methods of performing this procedure and the advantages and disadvantages of using one method over the other.

We will start by describing our three dependency extraction methods: Understand, srcML and Include [5], followed by a comparison of each method. Before performing the comparison we needed to have each file in the same format. It was suggested by professor Zhen Ming (Jack) Jiang to first have the files in ".ta" format to make the comparison easier since we already had a perl script from lab "transformUnderstand.pl" which converted a ".csv" file to the ".ta" format. The Understand method by default outputs in the ".csv" format while srcML and Include do not. To correct this we simply used excel to alter the srcML and Include output files into the ".csv" format, and then ran the perl script "transformUnderstand.pl" to obtain the ".ta" files for comparisons.

2.0 Dependency Extraction Techniques

2.1 Understand

Understand is a professional tool used to extract dependencies between files in large software systems. It is designed so that it helps to maintain and understand the large system code. Understand supports many legacy and modern programming languages like: Fortran, Ada, COBOL, C, Java, etc. Understand finds dependencies based on variety of things such as inheritance, import statements ,function calls and parameters, macro usage and many others. After the code is analyzed, Understand exports the results to a csv (Comma Separated File) file to be assessed with excel or any other tool. Understand is able to show the lines of code where dependencies occurred and which function call based dependencies exist using its dependency browser. Understand may need approximately 15 - 20 min to analyze code dependending on the hardware it's used on. Although MySQL contains java code we only considered C, C#, and C++ source code files, as these were the only files considered during assignment 2.

2.2 Include

A simple Java program developed by our team to retrieve import based dependencies found within the MySQL system. It goes through MySQL's source folder and searches all .c and .h files for #include statement, then checks if the included file exists in the MySQL project. If it exists, then a dependency is created. This makes sure that no standard library headers are counted as dependencies. The Include extraction process can be summarized in the following steps:

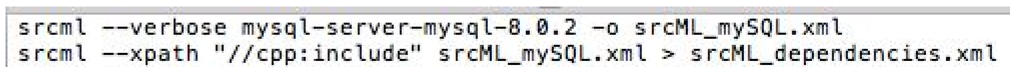
1. Search the MySQL project and add all header files with extensions .h, and .hpp to a list.
2. Recursively search through the MySQL source folder and scan each file within it. If the file is a directory, the function calls itself again to scan all entities in that directory. If the entity is a c,cc,cpp,h,or hpp file, then it step 3.
3. Scan the file for #include statements i.e #include <table.h> and extract the file name between "< >" check if it exists in the header list.
4. If it exists, then this file depends on that header, and write that dependency in the output file.

The script takes approximately 5 - 6 min to finish depends on the hardware. The drawbacks for using the include script is that it only considers #include statements. A file can depend on another through many ways such as function calls. Another drawback in using Include is that it doesn't consider #include statement that contain comments in them such as #include "<mysqld_error.h> /* to check server error codes */". this was caused by a bug in the code.

2.3 srcML

srcML is a free tool which transforms C, C++, C# and Java code into the srcML format, which is an XML representation of source code [1]. srcML is useful for analyzing and exploring source code as it can generate an XML file that maintains the property of the original source code. The files generated by srcML can be parsed using tools such as XPath. Tools which apply to the XML format can also be used on srcML code.

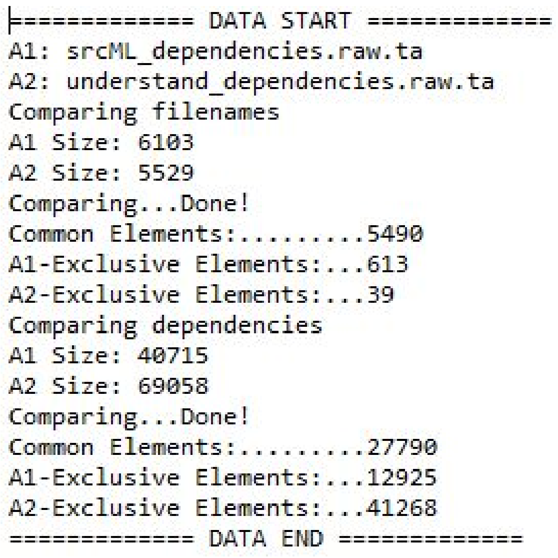
We ran srcML on the MySQL source code directory in order to generate a XML representation of the code, and used an XPath query to find all dependencies:



```
srcml --verbose mysql-server-mysql-8.0.2 -o srcML_mySQL.xml
srcml --xpath "//cpp:include" srcML_mySQL.xml > srcML_dependencies.xml
```

3.0 Dependency Comparison Processes

The dependencies were compared using using a java program called AnalyzeDependencies. It does all of the quantitative analysis and produced an easy to read output file with all of the dependency statistics from each technique.



```
============= DATA START =============
A1: srcML_dependencies.raw.ta
A2: understand_dependencies.raw.ta
Comparing filenames
A1 Size: 6103
A2 Size: 5529
Comparing...Done!
Common Elements:.........5490
A1-Exclusive Elements:...613
A2-Exclusive Elements:...39
Comparing dependencies
A1 Size: 40715
A2 Size: 69058
Comparing...Done!
Common Elements:.........27790
A1-Exclusive Elements:...12925
A2-Exclusive Elements:...41268
============= DATA END =============
```

Figure 1: Sample output from AnalyzeDependencies.java

The program works by first loading the dependencies into memory from the generated TA files. Once loaded, they can be easily compared by using the List and Collection data structures. The removeAll method will remove all common elements from the two lists and retainAll will remove all exclusive elements in the two lists. Doing the comparisons like this allows us to create new output files containing all dependencies between two files, which is needed later for the qualitative analysis.

4.0 Quantitative Analysis of Extraction Techniques

4.1 Understand Vs. Include

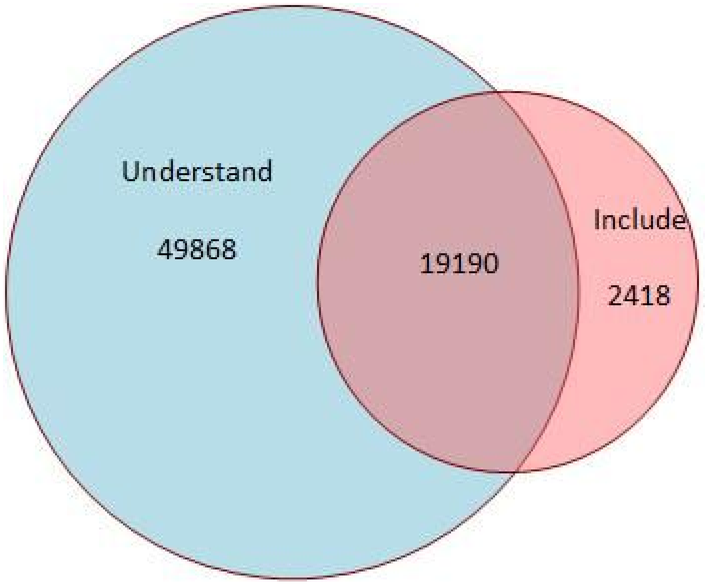


Figure 2: Understand vs Include venn diagram

It can be seen from **figure 2**, that the Include script has some shortcomings when compared to Understand. The number of common dependencies found was 19190, with 49868 exclusive to Understand and only 2418 exclusive to Include. This means that out of the total number of dependencies found, the Include script was only able to find 30% of them. This is a clear indication that Include script is missing something. Even considering that Understand is able to find dependencies beyond just the #include statements the Include script shouldn't fall so short.

4.2 Include Vs. srcML

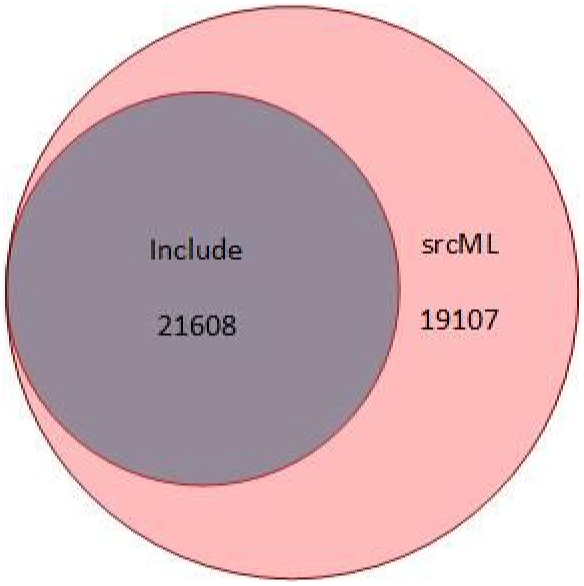


Figure 3: Include vs srcML venn diagram

Figure 3 is more useful for showing how much the include script is missing. Since srcML was configured to run similarly to Include, meaning it only considered the #include headers, they should theoretically produce the same results.

The total number of dependencies found by SrcML and Include is 40715, with 19107 exclusive to SrcML, 21608 common and 0 exclusive to Include. This shows us that Include was on the right track but there were more dependencies missed than the special ones that Understand looks for.

4.3 Understand Vs. srcML

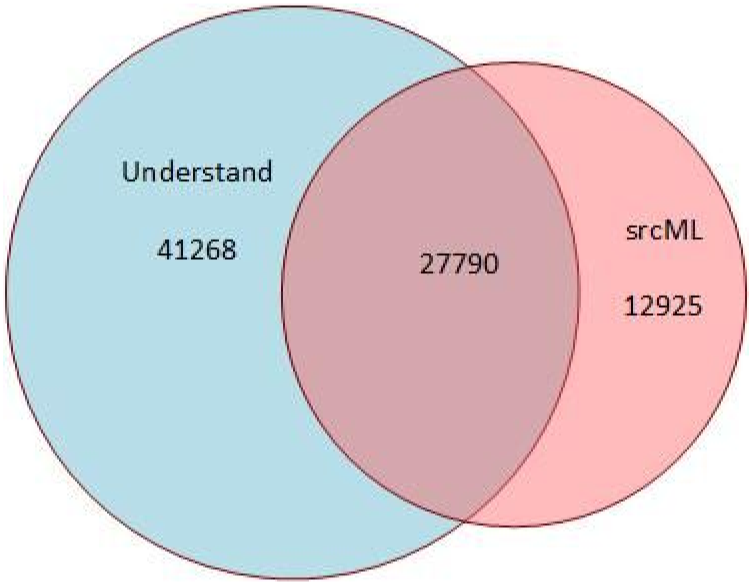


Figure 4: Understand vs. SrcML Venn Diagram

Figure 4 illustrates the differences with Understand and SrcML. The total number of dependencies found between the two is 81983, the highest of all the comparisons. Of the total number of dependencies, 41268 were exclusive to Understand and 12925 were exclusive to SrcML. A majority of the discrepancies between them are likely due to the fact that Understand looks at more than just the include headers.

6.0 Qualitative Analysis of Extraction Techniques

In order to do a feasible qualitative analysis of our extraction techniques we examined a portion of them using a sample size of randomly generated dependencies. In order to find our sample size we used the online calculator provided by Creative Research Systems [2], which given a confidence level of 95%, confidence interval of 5, and population of 81983 (Total dependencies), produced a sample size of 382. We then wrote a script “sample.java” which fetched 382 random dependencies and compared them to see what portion was contained in each method. Below is analysis of found discrepancies, and a discussion on the precision and recall results.

6.1 Understand Vs. Include

6.1.1 Dependencies found by understand only

Example 1:

mysql-server-mysql-8.0.2\vio\vioshm.cc → mysql-server-mysql-8.0.2\include\ my_io.h

- The reason for this dependency is that vioshm.cc uses a macro named SOCKET_ETIMEDOUT that is defined inside the my_io.h file
- Understand was able to get this dependency through macro usage even though the file vioshm.cc did not include the header my_io.h

Example 2:

mysql-server-mysql-8.0.2\sql\dd\dd_trigger.cc → dd_trigger.h

- The dependency created here because dd_trigger.cc uses the function

```
bool create_trigger(THD *thd, const ::Trigger *new_trigger,
                    enum_trigger_order_type ordering_clause,
                    const LEX_CSTRING &referenced_trigger_name)
```

Which is defined in this header `#include "trigger.h"     // Trigger.` Even though the header file was included in the file dd_trigger.cc, our include script failed to find this dependency because of the bug we talked about earlier in this report where it does not consider #includes with comments on them.

6.1.2 Dependencies found by Include only

Example 1:

mysql-server-mysql-8.0.2\client\mysql.cc→mysql-server-mysql-8.0.2\extra\libedit\editline\ readline.h

- It was found that mysql.cc doesn't use any function from readline.h therefore its inclusion there is redundant.
- Understand was smart enough not to depend only on includes but also searched if it was actually being used.

Example 2:

mysql-server-mysql-8.0.2\sql\log_event.cc → sql_lex.h

- It was found that log_event.cc doesn't use any functions from sql_lex.h therefore its inclusion there is redundant.
- Include script considered it as a dependency because of the include statement only as it did not search if it was actually being used.
- Understand was smart enough not to depend only on includes but also searched if it was actually being used.

6.1.3 Dependencies found in both

Msql-server-mysql-8.0.2\client\base\abstract_connection_program.cc→

mysql-server-mysql-8.0.2\client\base\abstract_connection_program.h

- found in both because of Include statement (case of the include script and understand) as well as function usage (case of the Understand).

6.2 Include Vs. srcML

6.2.1 Dependencies found by Include only

None because srcML contained everything our Include script was able to find.

6.2.2 Dependencies found by srcML only

There were many dependencies found by srcML only and that is because of the bugs in the Include script and because srcML included the standard system libraries as well. This sample output below is the result of running the command FC on windows to compare the srcml file and the Include script file

```
***** Include_Dependencies.txt

client\base\simple_option.cc → mysql-server-mysql-8.0.2\client\base\simple_option.h

client\base\simple_option.h → mysql-server-mysql-8.0.2\include\my_getopt.h
```

***** SRCML_DEPENDENCIES.TXT

Client\base\simple_option.cc → mysql-server-mysql-8.0.2\client\base\simple_option.h

client\base\simple_option.cc → stddef.h - Standard System library Include script does not take Standard System library headers

client\base\simple_option.h → string - Include Script considered only headers with.h extensions

client\base\simple_option.h → mysql-server-mysql-8.0.2\include\my_getopt.h

As it can be seen from the output srcML did count the header stddef.h as a dependency although it did not exist in the MySQL project because it is a C standard system library.

6.2.3 Dependencies found by srcml and Include

All dependencies found by Include were found by srcML but not the otherway around.

6.3 Understand Vs. srcML

6.3.1 Dependencies found by Understand only

Example 1:

mysql-server-mysql-8.0.2\storage\ndb\src\mgmsrv\InitConfigFileParser.cpp → mysql-server-mysql-8.0.2\storage\ndb\src\common\util\require.c

- Although require.c isn't directly imported by InitConfigFileParser.cpp , Understand wa still able to catch the indirect dependencies between these files.
- InitConfigFileParser.cpp uses a “require” feature that is implemented in different ways in a file called ndb_global.h.
- ndb_global uses a version of the “require” feature that is implemented in require.c

Example 2:

Storage\innobase\sync\sync0debug.cc → include\my_inttypes.h

- my_inttypes.h defines many min and max int values as well as TRUE/FALSE. These definitions are then used by sync0debug.cc

6.3.2 Dependencies found by srcML only

mysql-server-mysql-8.0.2\storage\innobase\row\row0upd.cc →
mysql-server-mysql-8.0.2\extra\rapidjson\include\rapidjson\msinttypes\inttypes.h

- This is an example of one of the many unneeded dependencies captured by the srcML method.
- Because of the nature of the srcML extraction method, this dependency was picked up merely because the inttypes.h file was included in the header. Features from it werent actually used.
- Understand was smart enough to see this and exclude it from its generated dependencies.

6.3.3 Dependencies found by Understand and srcML

Storage\innobase\handler\i_s.cc → storage\innobase\include\btr0btr.h

- Functionality from btr0btr.h is used in i_s.cc. Namely btr_page_get_index_id(page).

storage\ndb\src\kernel\blocks\trix\Trix.cpp →
storage\ndb\include\kernel\signaldata\WaitGCP.hpp

6.4 Precision and Recall

Precision and Recall is a statistical performance measurement which essentially estimates the usefulness and completeness of your data. More precisely, Precision is defined as the proportion of selected items that are relevant, and Recall is defined as the proportion of relevant items that are selected. Using the terms true positive(TP), false positive(FP), and false negative(FN), we can define Precision and Recall mathematically as follows:

$$\text{Precision} = \frac{tp}{tp + fp} \qquad \text{Recall} = \frac{tp}{tp + fn}$$

Below we examine the quality of the Include and srcML dependency extraction methods by using Precision and Recall, and assuming Understand acts as an oracle(dependencies extracted by Understand are all correct and present). The following tables lists one instance of our "sample.java" script [4] which will be used for the quality examination to come.

Sample Size: 382

Method	Understand	srcML	Include

Dependencies In Sample	321	204	150
Percentage In Sample	84%	39%	53%

6.4.1 srcML Calculations

When examining srcML we define the variables used to calculate Precision and Recall as follows:
TP = Dependencies from Sample also present in Understand = 321
FP = Dependencies from Sample also present in Include = 150
FN = Dependencies from Sample not present in Understand = 61

Precision = TP/(TP + FP) = 321/(321 + 150) = 68%
Recall = TP/(TP + FN) = 321(321 + 61) = 84%

6.4.2 Include Calculations

When examining Include we define the variables used to calculate Precision and Recall as follows:
TP = Dependencies from Sample also present in Understand = 321
FP = Dependencies from Sample also present in srcML = 204
FN = Dependencies from Sample not present in Understand = 61

Precision = TP/(TP + FP) = 321/(321 + 204) = 61%
Recall = TP/(TP + FN) = 321(321 + 61) = 84%

6.4.3 Analysis of Results

Quality of Extraction Methods

Method	srcML	Include
Precision	68%	61%
Recall	84%	84%

As we can see from the "Quality of Extraction Methods" table above, srcML has a slightly higher precision(68%) than Include(61%). This is expected and due to the fact that our implementation of the Include extraction process would emit some Include statements that were surrounded by strangely placed comments. In general the precision of srcML should be a bit higher, however, our implementation only considers "include" statements, but as we later learned srcML is also capable of searching for dependencies at the function level and method level, which would increase its precision. When examining the Recall of each extraction method, we see that srcML and Include both have the same Recall of 84%. This is also expected as srcML extracts marginally more dependencies than Include(approx. 7000), which would not affect the completeness of the representation of dependencies in the sample size.

6.5 F-Measure

The F-Measure, also know as F_1 Score combines Precision and Recall to measure a test's accuracy. The F-Measure is the harmonic mean of Precision and Recall or otherwise known as the average of when the two of them are close. The F-Measure best value is at 100%(perfect precision and recall), and worst at 0%. Mathematically it is defined as follows:

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Using our calculations from above we get the following F-Measures:

F-Measures

Method	srcML	Include
F-Measure	75%	70%

From the F-Measures table we see that srcML has an accuracy of 75% extracting dependencies, while Include has an accuracy of 70%.

7.0 Risks and Implications

Excluding the Understand dependency extraction the major risk we incurred was only considering dependencies based on "include" statements. As the assignment progressed we learned that dependencies also exist at the function level, method level, and via inheritance. Due to this some dependencies were missing from srcML and Include. Concerning the sample size, we chose the confidence interval arbitrarily to be 5. Choosing a higher interval reduces the sample size, while choosing a lower one increases it. Further statistical analysis would be required to find the optimal interval. Also, when generating the random sample the java function "rand()" was used. By default "rand()" uses a normal distribution to generate numbers, however it is not certain that our data(dependencies) were distributed normally. This could mean that the dependencies chosen in our sample size were actually chosen pseudo-random where dependencies within two standard deviations of the mean were favoured over outliers. Further statistical analysis would be required to confirm if our dependencies were distributed normally. Another potential risk lies with the comparison process of AnalyzeDependencies.java [6], where all dependencies are loaded into memory first. Doing this for a very large system could cause crashes due to running out of memory. It would have been better to analyze in smaller portions.

8.0 Lessons Learned

Unfortunately late in the project timeline we learned that MySQL did actually contain some java files and not only C, C#, and C++ as we originally assumed. Although theses files were mostly dealing with connectors, none of our extraction techniques considered them. We also learned that srcML was more powerful than originally thought. It could not only search for dependencies based on "include" statements, but could also use XPath queries to search for dependencies at the function level, method level, and via inheritance, similar to Understand. During the assignment we also learned that our communication could have been better. We had divided the assignment into individual tasks for each team member, and when it was time to compare the ".ta" files for dependencies it was not known that the files were sorted alphabetically. The "diff" command in unix would have been able to compare the two files easily, without the need of a java program.

9.0 References

[1]-http://www.srcml.org/about.html
[2]-https://www.surveysystem.com/sscalc.htm
[3]-https://github.com/hashim93a/eecs4314MYSQL/tree/master/Assignments/Assignment3/Scripts
[4]-https://github.com/hashim93a/eecs4314MYSQL/blob/master/Assignments/Assignment3/Scripts/qualitative%20analysis/sample.java
[5]-https://github.com/hashim93a/eecs4314MYSQL/blob/master/Assignments/Assignment3/Scripts/Include.java
[6]-https://github.com/hashim93a/eecs4314MYSQL/blob/master/Assignments/Assignment3/Scripts/AnalyzeDependencies.java