EECS 4314: Hadoop Dependency Extraction Report

Authors
Mina Zaki
Alexander Aolaritei
Dillon Bondarenko
Camillo John (CJ) D'Alimonte
Jeremi Boston
Daniel Nowak

November 18, 2016

Contents

1 Abstract

This report discusses three dependency extraction techniques. These three techniques are dependency extraction using Include directives, Understand, and srcML. Following the discussion of these three techniques there will be a quantitative and qualitative comparison on their extraction on the Hadoop source code. To help illustrate techniques, use cases will be provided.

2 Introduction

Three dependency extraction tools along with their results will be covered. Next, there will a quantitative and qualitative comparison of the extraction tools. Following the comparison, a demonstration of two of the dependency extraction techniques using use cases will be shown. Finally, a discussion of the limitations of the reported findings and the lessons learned from this exercise will be presented.

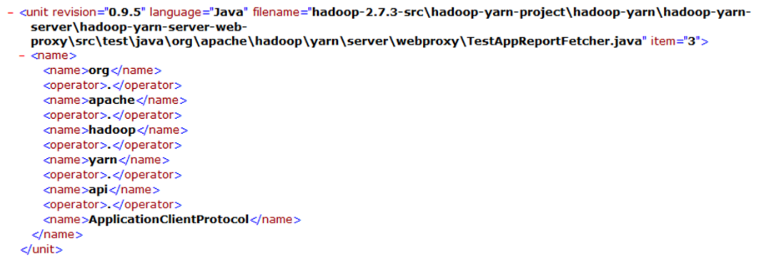
3 Dependency Extraction using Include Directives

```
// go through each of the files, however if they do not end with .java, then skip them

if (fileName.endsWith(".java")) {

   try (Stream<String> stream = Files.lines(Paths.get(fileName))) {

      // add all of the lines in the current file that contain
      // the word import org.apache. to the list, list

      list = stream.filter(line ->
          line.startsWith(``import org.apache.``))
              .collect(Collectors.toList());

   } catch (IOException e) {
      e.printStackTrace();
   }

   // for each line that contains import to a hadoop
       // class in the current file

   for (String l : list) {

      // output_line is the line, however it removes the
      // ``import `` from the beginning and the ';' from the end

      String output_line = l.toString().substring(7,
          l.toString().length() - 1);

        // add this line to the output file

      output.add(fileEntry.getName() + `` --> `` + output_line);
   }
```

This extraction method involves going through the Hadoop Source Code directory and finding all of the *.java* files. In each *.java* file, the lines which contain the word import are documented. This produces all of the lines that have the keyword import, however many of these lines (approximately 30000) are dependencies of the Java classes. To extract only the dependencies from one class in Hadoop to another, we must use only the lines that begin with *import org.apache.* Then each element in this list is converted into the TA format. This format is defined as $C \rightarrow D$, where C depends on D. In this case, the current file is C and the class it depends upon is D. Once completed, the lines are stored in an output file.

4 Dependency Extraction using srcML

SourceML, or srcML, is the third extraction method used in this assignment. It allows users to explore, analyze, and even manipulate source code through the use of the XML file it provides as the output. It is a lightweight parsing tool which enables users to extract specific metrics that are searched upon. srcML is made readily available at no cost. Given some arbitrary code file, the output of srcML is an XML document that does not modify the original file, but rather outputs it as an XML file that retains all the original's properties. It can be thought of as just another programming language. However, Hadoop as a whole is not just one block of code. The structure of Hadoop is divided into many directories: Java files, Shell files, as well as their own XML files. The Java files were the main focus points of interest in this project as the task was to recurse through each subdirectory and select only the Java files which were used. Once the Java files were fed into the srcML program with the desired query, the output of each was appended to an XML file formatted by srcML. An example of what the XML output looks like can be found in Figure 1.



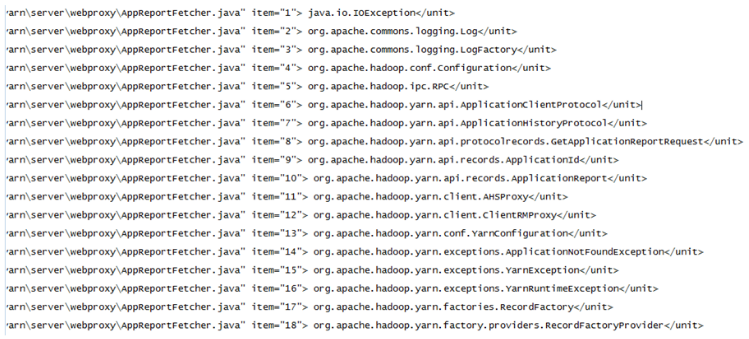
```
- <unit revision="0.9.5" language="Java" filename="hadoop-2.7.3-src\hadoop-yarn-project\hadoop-yarn\hadoop-yarn-
    server\hadoop-yarn-server-web-
    proxy\src\test\java\org\apache\hadoop\yarn\server\webproxy\TestAppReportFetcher.java" item="3">
  - <name>
      <name>org</name>
      <operator>.</operator>
      <name>apache</name>
      <operator>.</operator>
      <name>hadoop</name>
      <operator>.</operator>
      <name>yarn</name>
      <operator>.</operator>
      <name>api</name>
      <operator>.</operator>
      <name>ApplicationClientProtocol</name>
    </name>
  </unit>
```

Figure 1: XML file generated from srcML

4.1 Extracted File

Figure 1 shows which file was placed into srcML with the "filename" tag; the "item" tag determines which occurrence of the specified query it was, and the body of this block yields exactly what the searched term was. In this case, an import statement to *org.apache.hadoop.yarn.api.ApplicationClientProtocol* is the 3rd such item found in the file *TestAppReportFetcher.java*. So what does this mean? srcML has the neat little function in which you can parse through an XML file to further enhance your query.

Using the above knowledge, one can even output a more in depth query into a more human-readable format. A text document was used in this case to easily enumerate when an occurrence of the keyword is found, how many there were, and what exactly it was. Figure 2 shows the format of such a text file.



```
arn\server\webproxy\AppReportFetcher.java" item="1"> java.io.IOException</unit>
arn\server\webproxy\AppReportFetcher.java" item="2"> org.apache.commons.logging.Log</unit>
arn\server\webproxy\AppReportFetcher.java" item="3"> org.apache.commons.logging.LogFactory</unit>
arn\server\webproxy\AppReportFetcher.java" item="4"> org.apache.hadoop.conf.Configuration</unit>
arn\server\webproxy\AppReportFetcher.java" item="5"> org.apache.hadoop.ipc.RPC</unit>
arn\server\webproxy\AppReportFetcher.java" item="6"> org.apache.hadoop.yarn.api.ApplicationClientProtocol</unit>
arn\server\webproxy\AppReportFetcher.java" item="7"> org.apache.hadoop.yarn.api.ApplicationHistoryProtocol</unit>
arn\server\webproxy\AppReportFetcher.java" item="8"> org.apache.hadoop.yarn.api.protocolrecords.GetApplicationReportRequest</unit>
arn\server\webproxy\AppReportFetcher.java" item="9"> org.apache.hadoop.yarn.api.records.ApplicationId</unit>
arn\server\webproxy\AppReportFetcher.java" item="10"> org.apache.hadoop.yarn.api.records.ApplicationReport</unit>
arn\server\webproxy\AppReportFetcher.java" item="11"> org.apache.hadoop.yarn.client.AHSProxy</unit>
arn\server\webproxy\AppReportFetcher.java" item="12"> org.apache.hadoop.yarn.client.ClientRMProxy</unit>
arn\server\webproxy\AppReportFetcher.java" item="13"> org.apache.hadoop.yarn.conf.YarnConfiguration</unit>
arn\server\webproxy\AppReportFetcher.java" item="14"> org.apache.hadoop.yarn.exceptions.ApplicationNotFoundException</unit>
arn\server\webproxy\AppReportFetcher.java" item="15"> org.apache.hadoop.yarn.exceptions.YarnException</unit>
arn\server\webproxy\AppReportFetcher.java" item="16"> org.apache.hadoop.yarn.exceptions.YarnRuntimeException</unit>
arn\server\webproxy\AppReportFetcher.java" item="17"> org.apache.hadoop.yarn.factories.RecordFactory</unit>
arn\server\webproxy\AppReportFetcher.java" item="18"> org.apache.hadoop.yarn.factory.providers.RecordFactoryProvider</unit>
```

Figure 2: A second iteration through the srcML technique

As shown, this removes the clutter created from the series of <tag>'s while enumerating the file, item#, and target in a nice order that can be easily converted into the TA format. It was formed using the filename → target format; the item count was not necessarily a requirement for this format, however it was still a neat property to keep for measuring metrics. With this file and the output generated from the include technique, a comparison was made to find any differences among the two methods.

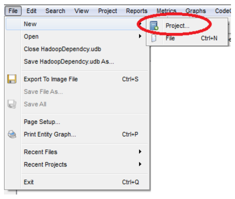
4.2 Advantages and Disadvantages

Each method used to extract their dependencies had their own advantages and disadvantages. One great tool you can use srcML for is that it can parse through a file for any keyword, string, regular expression in your source code. Since it also lists the number of the specified item found, any desired metric can be quantified. If the user wanted to know how many "if, then, else" branches were used throughout the source code or how many times the method "getUser()" was called?, srcML could accomplish all this. The generated XML file can be further parsed thereby making tasks such as "Determine the instances that an import statement was used, but NOT including *hadoop.yarn* possible. Once the XML file is generated, the tags can provide any other application that uses XML files with a wide array of functionality.

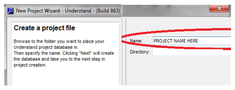
Some disadvantages including the inability to determine calls from an arbitrary *X.java* file to the user's designated input file (something accomplished through the Understand tool). The immense usage of <tags> throughout the XML document often clutter your file. In larger software systems such as Hadoop, the XML file generated is rather large and takes a considerable amount of time to fully load. As srcML is still in the development phase, its only supported languages are the C family and Java.

5 Dependency Extraction using Understand

Understand is a static analysis tool which is mainly used for the purpose of standards testing, metrics and source code comprehension. It is essentially used to maintain large source code bases, either legacy or new, and provide a better understanding of their structures. The way a user interacts with Understand is through a multi-language, cross platform, maintenance-oriented interactive development environment (IDE). This idea offers its services on source code in the following languages: C, C++, C#, Objective C/Objective C++, Ada, Java, Pascal/Delphi, COBOL, JOVIAL, VHDL, Fortran, PL/M, Python, PHP, HTML, CSS, JavaScript, and XML. The Understand software tool offers many features such as code knowledge, metrics and reports, graphing, standards testing, dependency analysis, an Editor and source code search. The most important of these features, at least in terms of this report, is the dependency analysis. The Understand software is capable of extracting the following dependencies: inheritance, imports, throws, implementation, Java annotations (@), object initialization and method calls. The way it extracts these dependencies is by examining every reference in the project until a dependency data structure is built for every file and architecture. This structure will include the references that caused the dependencies and the nature of such dependencies. Due to the amount of data being large, this is not calculated as you ask for a certain dependency relationship. Rather, all dependency information is calculated and cached, which results in quick exploration and browsing. Understand's dependency analysis provides many capabilities for those wishing to further understand their source code such as quick navigation of dependencies for project architectures and files and the creation of graphs for architectures and files. It can also export these dependency relationships to a spreadsheet in Excel (CSV) and it also offers a dependency browser which allows to explore every dependency and its attached information. The following are the steps taken, using Understand, to extract the dependencies in the Hadoop source code.



Step 1. Create a new project


New Project Wizard - Understand - (Build 863)

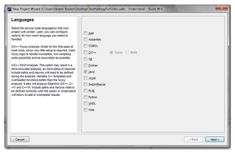
Create a project file

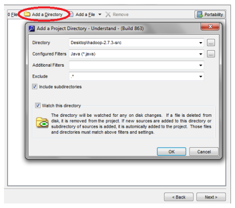
Browse to the folder you want to place your Understand project database in.
Then specify the name. Clicking "Next" will create the database and take you to the next step in project creation.

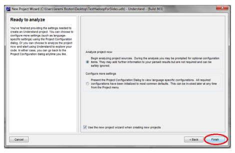
Name: PROJECT NAME HERE

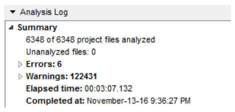
Directory:

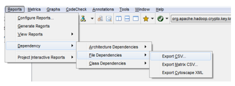
Step 2. Name your project


Step 3. Select the source code language

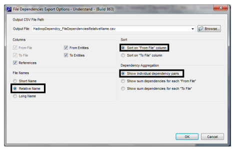

Step 4. Add the source code directory

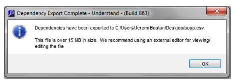

Step 5. Run the Understand Project Analysis


```
Analysis Log
Summary
    6348 of 6348 project files analyzed
    Unanalyzed files: 0
    Errors: 6
    Warnings: 122431
    Elapsed time: 00:03:07.132
    Completed at: November-13-16 9:36:27 PM
```
Step 6. When completed it will display an analysis log



Step 7. Export the dependencies report to CSV


Step 8. Select the dependencies options for the export


Dependency Export Complete - Understand - (Build 863)

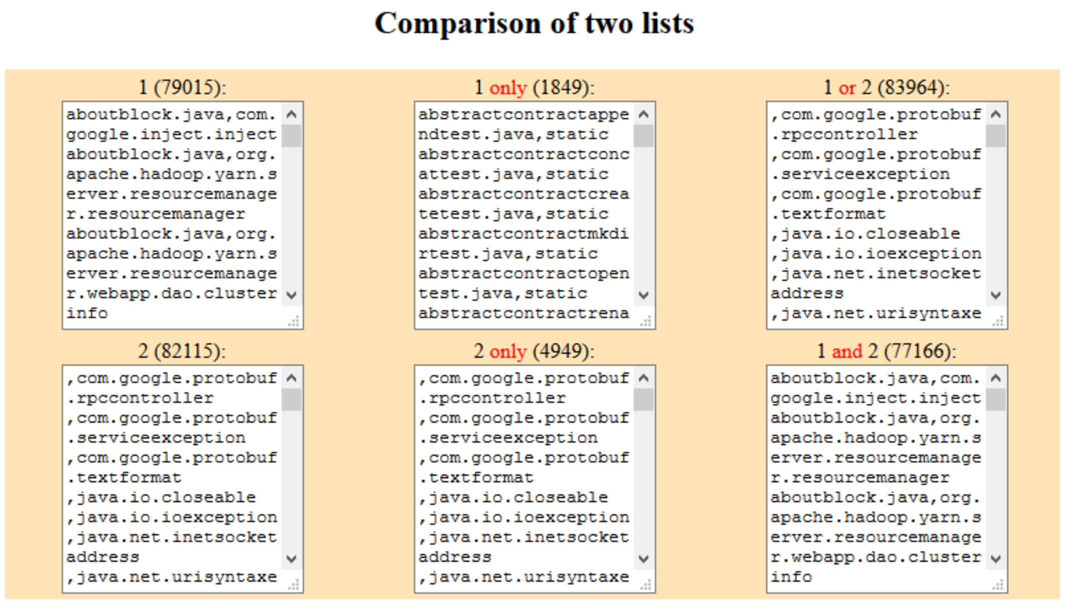
Dependencies have been exported to C:/Users/Jeremi Boston/Desktop/poop.csv.

This file is over 15 MB in size. We recommend using an external editor for viewing/editing the file

Step 9. You are finished extracting the dependencies of your source code to CSV

6 Quantitative Results

The following analysis used three separate programs - Understand, srcML, and Include - to discover and present the implemented dependencies within Hadoop. While using Understand, a total of 83,000 class dependencies were found along with 62,000 file dependencies. With srcML there were 87,000 dependencies discovered, and finally Include was able to find nearly 84,000 file dependencies. These dependencies were then taken and stored into files to be further formatted. To allow for easy comparisons, all the files were formatted using the same process. Using Figure 3 as a reference, when comparing the results from the srcML dependency list (number 1) and the Include dependency list (number 2), there are 77166 dependencies that are common in both lists. This leaves 1849 dependencies which are unique to Include while leaving 4949 results unique to the srcML method. It was surprising to find so many unique results as both methods searched through the same Hadoop source code. However, a very significant portion of the unique dependencies seem to be from certain test files ranging from test-shells, file checkers, web-services, as well as from the Hadoop distributed file server. There also appears to be a number of unique dependencies shown to be specific import statements that were possibly not caught by one of the methods used for extraction. Lastly, the number of dependencies found and the number listed in Figure 3 are off by a small amount. This is most likely due to the program not being able to handle the large input list sizes; such input sizes would develop into becoming one of the limitations of the report.


Comparison of two lists

1 (79015):
```
aboutblock.java,com.
google.inject.inject
aboutblock.java,org.
apache.hadoop.yarn.s
erver.resourcemanage
r.resourcemanager
aboutblock.java,org.
apache.hadoop.yarn.s
erver.resourcemanage
r.webapp.dao.cluster
info
```

1 only (1849):
```
abstractcontractappe
ndtest.java,static
abstractcontractconc
attest.java,static
abstractcontractcrea
tetest.java,static
abstractcontractmkdi
rtest.java,static
abstractcontractopen
test.java,static
abstractcontractrena
```

1 or 2 (83964):
```
,com.google.protobuf
.rpccontroller
,com.google.protobuf
.serviceexception
,com.google.protobuf
.textformat
,java.io.closeable
,java.io.ioexception
,java.net.inetsocket
address
,java.net.urisyntaxe
```

2 (82115):
```
,com.google.protobuf
.rpccontroller
,com.google.protobuf
.serviceexception
,com.google.protobuf
.textformat
,java.io.closeable
,java.io.ioexception
,java.net.inetsocket
address
,java.net.urisyntaxe
```

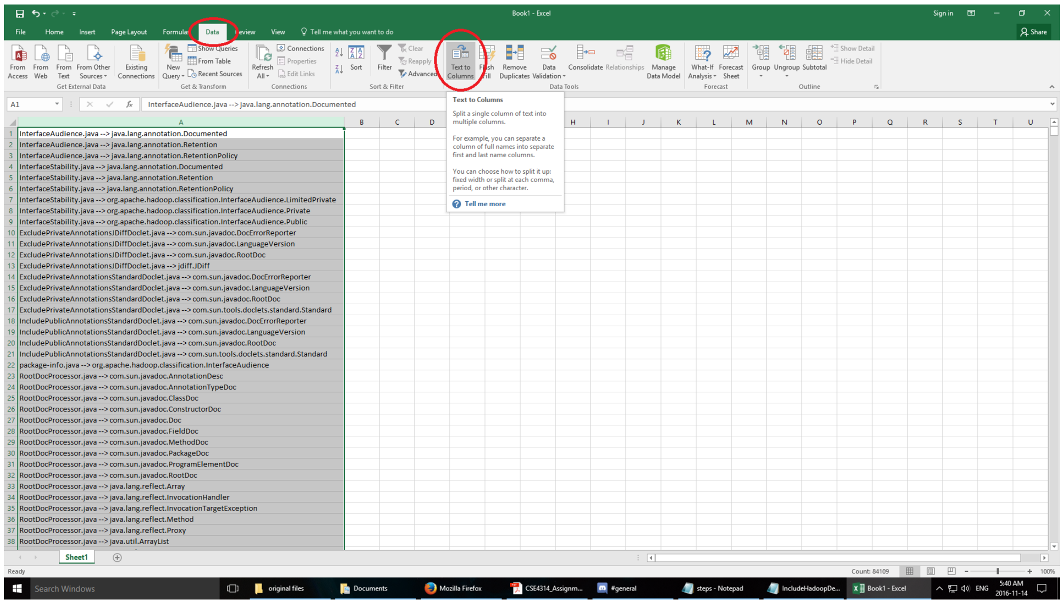
2 only (4949):
```
,com.google.protobuf
.rpccontroller
,com.google.protobuf
.serviceexception
,com.google.protobuf
.textformat
,java.io.closeable
,java.io.ioexception
,java.net.inetsocket
address
,java.net.urisyntaxe
```

1 and 2 (77166):
```
aboutblock.java,com.
google.inject.inject
aboutblock.java,org.
apache.hadoop.yarn.s
erver.resourcemanage
r.resourcemanager
aboutblock.java,org.
apache.hadoop.yarn.s
erver.resourcemanage
r.webapp.dao.cluster
info
```

Figure 3: The program used to compare two lists of dependencies from srcML and Include where list 1 is from srcML and list 2 from Include

6.1 Quantitative Process

This is a brief step by step description on how the dependency data sets from the 3 different methods were compared. The first step consisted of gathering all the data by outputting the information from the three tools into a text file which may then be used for analysis. Following the data retrieval, one must format the data to provide an easy and efficient way of comparing the two lists against each other. The formatting performed in this report entailed the use of Excel to get rid of unwanted information and neatly compile the lists together. As seen on Figure 4, the list was separated using the text to column feature in Excel located in the Data tab. This feature allows one to organize data by organizing the list into separate columns through the use of a delimiter. The case below in Figure 4 made use of a space character as a delimiter to separate the data. After the data has been organized into their own columns the unwanted data such as the arrows were removed. Now the spreadsheet contains two lists, the first column A consists of a file which is dependent on the file in column B. Finally, the data is saved as a *.csv* file which is a comma delimited file used to separate the two dependencies. The same process is repeated for the remaining data from the other two tools. The reasoning for the comma delimited format rather than the Sub-Tuple language format, is due to a few unexpected problems which occurred when trying to output the data into the specified format straight from the tool.



Text to Columns

Split a single column of text into multiple columns.

For example, you can separate a column of full names into separate first and last name columns.

You can choose how to split it up: fixed width or split at each comma, period, or other character.

A1: InterfaceAudience.java --> java.lang.annotation.Documented

	A
1	InterfaceAudience.java --> java.lang.annotation.Documented
2	InterfaceAudience.java --> java.lang.annotation.Retention
3	InterfaceAudience.java --> java.lang.annotation.RetentionPolicy
4	InterfaceStability.java --> java.lang.annotation.Documented
5	InterfaceStability.java --> java.lang.annotation.Retention
6	InterfaceStability.java --> java.lang.annotation.RetentionPolicy
7	InterfaceStability.java --> org.apache.hadoop.classification.InterfaceAudience.LimitedPrivate
8	InterfaceStability.java --> org.apache.hadoop.classification.InterfaceAudience.Private
9	InterfaceStability.java --> org.apache.hadoop.classification.InterfaceAudience.Public
10	ExcludePrivateAnnotationsJDiffDoclet.java --> com.sun.javadoc.DocErrorReporter
11	ExcludePrivateAnnotationsJDiffDoclet.java --> com.sun.javadoc.LanguageVersion
12	ExcludePrivateAnnotationsJDiffDoclet.java --> com.sun.javadoc.RootDoc
13	ExcludePrivateAnnotationsJDiffDoclet.java --> jdiff.JDiff
14	ExcludePrivateAnnotationsStandardDoclet.java --> com.sun.javadoc.DocErrorReporter
15	ExcludePrivateAnnotationsStandardDoclet.java --> com.sun.javadoc.LanguageVersion
16	ExcludePrivateAnnotationsStandardDoclet.java --> com.sun.javadoc.RootDoc
17	ExcludePrivateAnnotationsStandardDoclet.java --> com.sun.tools.doclets.standard.Standard
18	IncludePublicAnnotationsStandardDoclet.java --> com.sun.javadoc.DocErrorReporter
19	IncludePublicAnnotationsStandardDoclet.java --> com.sun.javadoc.LanguageVersion
20	IncludePublicAnnotationsStandardDoclet.java --> com.sun.javadoc.RootDoc
21	IncludePublicAnnotationsStandardDoclet.java --> com.sun.tools.doclets.standard.Standard
22	package-info.java --> org.apache.hadoop.classification.InterfaceAudience
23	RootDocProcessor.java --> com.sun.javadoc.AnnotationDesc
24	RootDocProcessor.java --> com.sun.javadoc.AnnotationTypeDoc
25	RootDocProcessor.java --> com.sun.javadoc.ClassDoc
26	RootDocProcessor.java --> com.sun.javadoc.ConstructorDoc
27	RootDocProcessor.java --> com.sun.javadoc.Doc
28	RootDocProcessor.java --> com.sun.javadoc.FieldDoc
29	RootDocProcessor.java --> com.sun.javadoc.MethodDoc
30	RootDocProcessor.java --> com.sun.javadoc.PackageDoc
31	RootDocProcessor.java --> com.sun.javadoc.ProgramElementDoc
32	RootDocProcessor.java --> com.sun.javadoc.RootDoc
33	RootDocProcessor.java --> java.lang.reflect.Array
34	RootDocProcessor.java --> java.lang.reflect.InvocationHandler
35	RootDocProcessor.java --> java.lang.reflect.InvocationTargetException
36	RootDocProcessor.java --> java.lang.reflect.Method
37	RootDocProcessor.java --> java.lang.reflect.Proxy
38	RootDocProcessor.java --> java.util.ArrayList

Count: 84109

Figure 4: A view of the data in Excel and the method used to format the given data

With the newly created files, an online comparison tool was used to compare the data in order to look for unique and overlapping entries. Instead of manually going through all the overlapping entries, a sample size calculator was used to figure out the number of samples needed to sufficiently represent the data. The sample size calculator used a confidence level of 95%, with a confidence level of 5, and a population size of 78,000 (rounded value) entries. This returned a result of 382 random samples needed to represent the overlapping data set. The overlapping entries were once again placed into excel which was used to randomly sort the dependencies by using a random number generator. After sorting, the first 382 entries were retrieved for further analysis.

7 Qualitative Results

7.1 Precision and Recall

Precision and recall are the most basic measures used in evaluating the effectiveness of search strategies. Precision is the ratio of the number of relevant records retrieved to the total number of irrelevant and relevant records retrieved. Recall is the ratio of the number of relevant records retrieved to the total number of relevant records in the database. Both measurements are usually expressed as a percentage. [1]

Since a sample size of 382 has been determined to be the most suitable survey for the comparison, let us consider a total of 382 dependencies that will be analyzed by both the Include and srcML tools. We define "correct" dependencies those that are shared amongst the analysis tools while "incorrect" dependencies were unique to a specific tool, either Include or srcML. The number of dependencies analyzed by both tools in the 382 dependency sample size is proportional to the number analyzed in terms of the entire 83,964 dependencies (i.e. 0.9765 of dependencies were common amongst the tools).
Include:
382 dependencies analysed, 365 dependencies shared in common (correct), 9 unique dependencies (incorrect)
PRECISION: $\frac{365}{9+365} * 100\% = 0.9756$
RECALL: $\frac{365}{8+365} * 100\% = 0.9786$
srcML:
382 dependencies analysed in total, 365 dependencies analyzed using srcML in common (correct), 8 unique dependencies (incorrect)
PRECISION: $\frac{365}{8+365} * 100\% = 0.9786$
RECALL: $\frac{365}{9+365} * 100\% = 0.9756$

7.2 Venn Diagram

The following Venn Diagram highlights the unique and shared dependencies between the Include and srcML extraction methods. In all, there are a combined 83,964 dependencies extracted between the two methods.

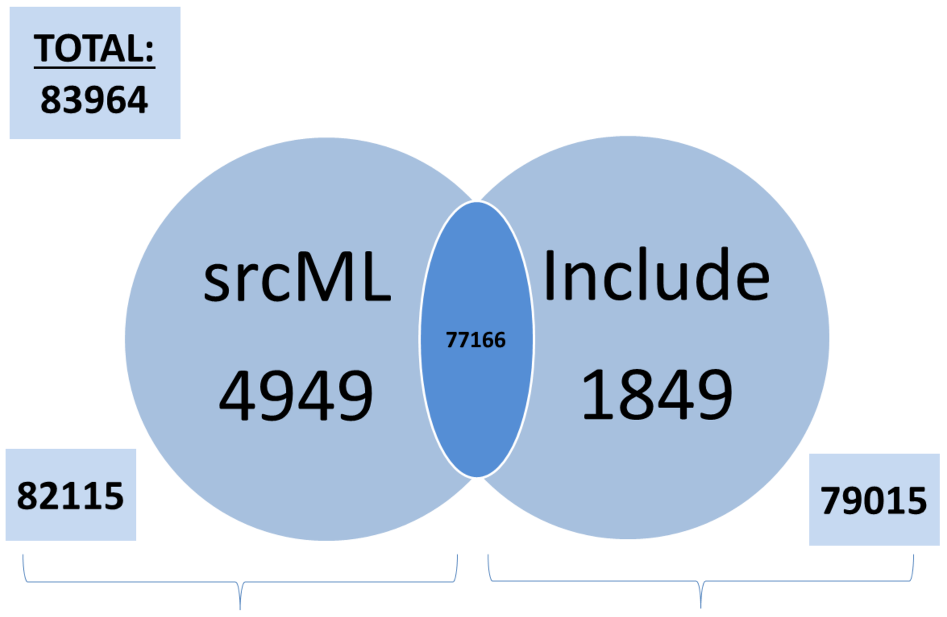


Figure 5: Unique & Shared Dependencies - srcML & Include

8 Use Cases

8.1 Understand Use Cases

Understand allows users to export metrics about dependencies between files, classes, and architectures in a CSV file format. The use case for extracting dependencies using Understand can be better observed in the sequence diagram shown in Figure 6. The first phase is to create a new project. This is done by clicking the File tab followed by New Project. During this phase the project name, directory, language, and source directory is configured. In this case, the source directory would be the Hadoop source code. After these configurations are set, Understand will be ready to begin analyzing the project. The next phase is dependency exportation. This can be done by clicking Reports, Dependency, and then choosing between exporting dependencies for architecture, files, or classes. Exporting architecture dependencies as a CSV will result in a content structure as shown in Figure 7 where the nodes in column A are dependent on the nodes displayed in column B. Similarly, exporting file/class dependencies will create a CSV file where the files/classes appearing in column A are dependent on the files/classes appearing in column B. [2]

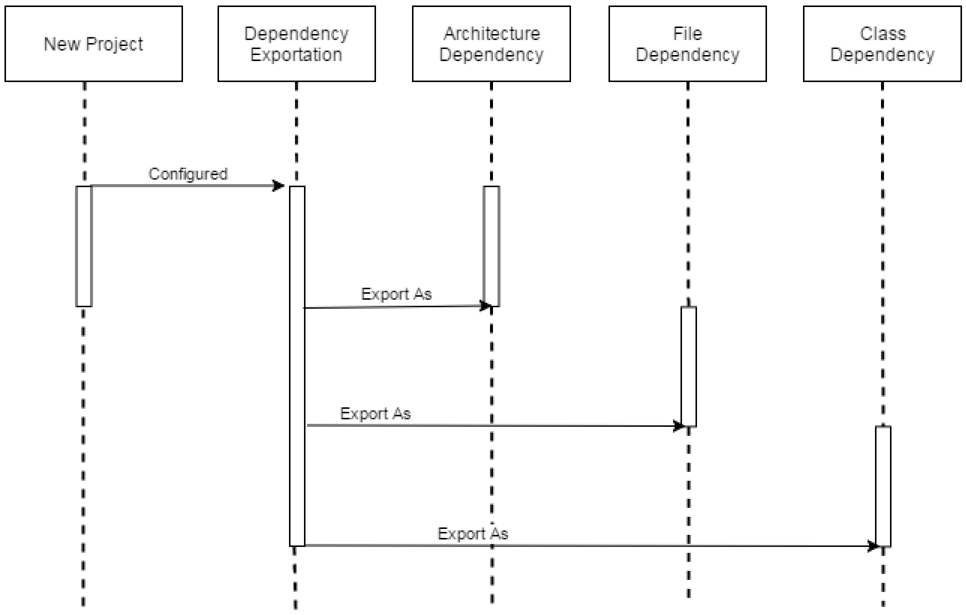
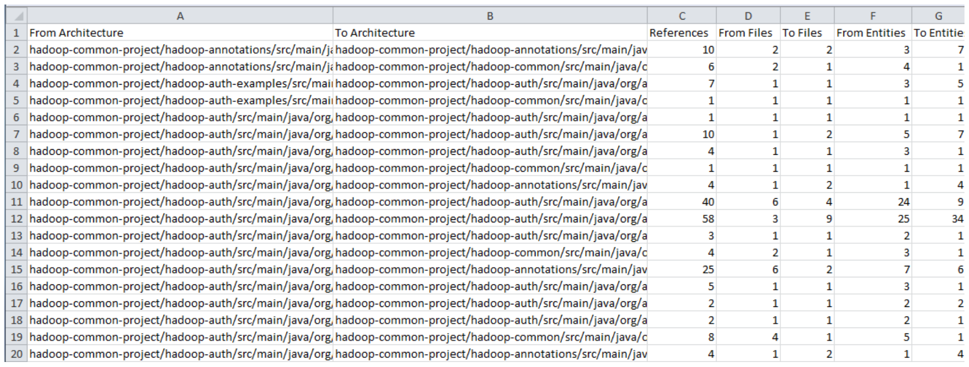


Figure 6: Sequence Diagram for the Understand Extraction Method

8.2 srcML Use Cases

There are other alternatives to extracting dependencies, one of them is srcML. The technique used to extract dependencies using srcML can be better observed in the state diagram shown in Figure 8. The first step is to execute the srcml command on the target file/directory within the operating systems command line. The srcml command will convert the file/directory into srcml format which is just the source code with tags identifying different syntactic parts of the source code. Afterwards, an XPath query is performed on the result obtained in the previous step. XPath queries allow users to perform syntactic and hierarchical searches on the source code. For example, an XPath query searching for the "import tag can be executed on the result returned from the srcml command. This will return a new result containing lines with the import tag. This result along with further



	A	B	C	D	E	F	G
1	From Architecture	To Architecture	References	From Files	To Files	From Entities	To Entitie
2	hadoop-common-project/hadoop-annotations/src/main/ja	hadoop-common-project/hadoop-annotations/src/main/jav	10	2	2	3	7
3	hadoop-common-project/hadoop-annotations/src/main/ja	hadoop-common-project/hadoop-common/src/main/java/c	6	2	1	4	1
4	hadoop-common-project/hadoop-auth-examples/src/mai	hadoop-common-project/hadoop-auth/src/main/java/org/a	7	1	1	3	5
5	hadoop-common-project/hadoop-auth-examples/src/mai	hadoop-common-project/hadoop-common/src/main/java/c	1	1	1	1	1
6	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-auth/src/main/java/org/a	1	1	1	1	1
7	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-auth/src/main/java/org/a	10	1	2	5	7
8	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-auth/src/main/java/org/a	4	1	1	3	1
9	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-common/src/main/java/c	1	1	1	1	1
10	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-annotations/src/main/jav	4	1	2	1	4
11	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-auth/src/main/java/org/a	40	6	4	24	9
12	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-auth/src/main/java/org/a	58	3	9	25	34
13	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-auth/src/main/java/org/a	3	1	1	2	1
14	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-common/src/main/java/c	4	2	1	3	1
15	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-annotations/src/main/jav	25	6	2	7	6
16	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-auth/src/main/java/org/a	5	1	1	3	1
17	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-auth/src/main/java/org/a	2	1	1	2	2
18	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-auth/src/main/java/org/a	2	1	1	2	1
19	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-common/src/main/java/c	8	4	1	5	1
20	hadoop-common-project/hadoop-auth/src/main/java/org/	hadoop-common-project/hadoop-annotations/src/main/jav	4	1	2	1	4

Figure 7: CSV Content Structure

XPath queries can be used for the dependency exportation result.

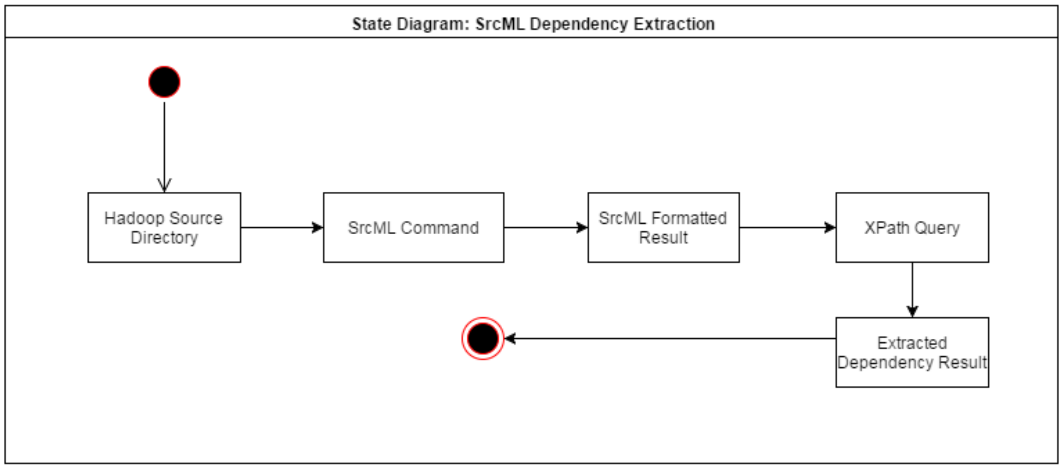

State Diagram: SrcML Dependency Extraction
Hadoop Source Directory
SrcML Command
SrcML Formatted Result
XPath Query
Extracted Dependency Result

Figure 8: State Diagram for the srcML Extraction Method

9 Lessons Learned

Throughout the process of the assignment, various techniques of understanding large systems were learned. These techniques include the use of tools to extract dependencies from file systems for analysis, how to pre-process data for easier analysis, and also how to decrease the amount of data to analyze while still maintaining a representative sample size of the data set. Not only does this allow one to better understand the system, but it also provides a method to search for unwanted design implementations, such as certain classes that should not be accessible to one another. When faced with the task of analyzing a new system, these skills will provide a software engineer with the methods needed to fully comprehend the system design.

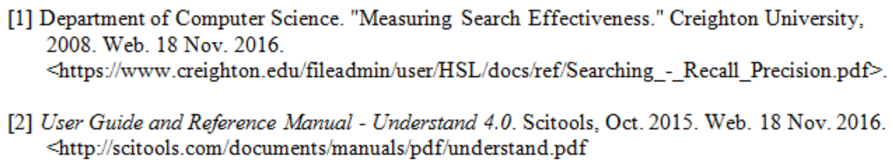
10 Limitations of Reported Findings

In the comparison process a random sampling was taken, the comparison may not be representative of the entire system. Another problem encountered is the online list comparator. Since the list of dependencies were so large it was not able to hold the full list of dependencies losing approximately 6% of the dependencies found.

11 Conclusion

In conclusion, there are many different ways to extract dependencies from a system. As shown in the comparison, they are not all equal and produce different results. In a system as complex as Hadoop, it is unrealistic that any two dependency extraction techniques yield exactly the same results. Comparing the three tools Understand, Include, and srcML the most in depth tool is Understand. This tools allows one to choose between the extraction of class dependencies or file dependencies, along with the ability to view pertinent information about functions, classes and variables. Understand provides an efficient and effective method of extracting dependencies of a systems and provides other important tools needed to fully comprehend the interworking components of the system.

12 Bibliography



[1] Department of Computer Science. "Measuring Search Effectiveness." Creighton University, 2008. Web. 18 Nov. 2016. <https://www.creighton.edu/fileadmin/user/HSL/docs/ref/Searching_-_Recall_Precision.pdf>.

[2] *User Guide and Reference Manual - Understand 4.0*. Scitools, Oct. 2015. Web. 18 Nov. 2016. <http://scitools.com/documents/manuals/pdf/understand.pdf