EECS 4314 - Advanced Software Engineering

Dependency Extraction Techniques

HBase

Nov 12, 2018

Daniel McVicar	213027479
Yahya Ismail	213235403
Daniel Fortunato	216796443
Adham El Shafie	212951018
Rafay Sheikh	213033451

Abstract

This report investigates dependency extraction techniques for large scale software projects. HBase is used as a case study to compare three different techniques: Understand, srcML, and a custom written dependency extractor. Each method produced a different set of dependencies and showed different strengths and weaknesses.

This report discusses the comparison process and how each extraction method was used to produce useful output data. Qualitative and quantitative analysis is performed on the results to determine which method is the most useful. The quantitative analysis reveals that Understand and srcML have many common dependencies as well as a significant disjoint. The custom written DependencyExtractor script produces the lowest number of dependencies. The qualitative analysis finds Understand to have the most precision in its output and a moderate return for recall.

Limitations and risks for the three different methods are discusses and checked against the data gathered from the HBase analysis. Lessons learned from this exercise are reviewed and the final conclusion is presented, which highlights the requirement for multiple dependency extraction techniques to overcome the flaws of each individual approach.

Introduction

Dependency extraction is essentially using the source code to see how the system works and what dependencies connect subsystems together. HBase is written almost completely in Java and so programs likes Understand are able to go through each class in the directories quickly and efficiently. The basic idea is a dependency occurs if class A uses class B and that can be seen if A imports B, or if A instantiates B, or if static fields or methods from B are used in A.

Many developers and engineers create high level models of the software they are examining in order to understand how the system works and come up with a solid architecture. It is almost impossible to recover the software architecture of a large and complex system by manually reading source code, as you simply can't go in the source files line by line. Having the dependencies extracted automatically allow developers to get rid of their assumptions as they are able to visualize and understand the software better, as well as free up time to work on more pressing issues. We saw a similar pattern when we went from conceptual or concrete architecture. This process of extraction is seen in practise in the architecture understanding process used by engineers when they compare and investigate how their high level models differ from the concrete architecture and using this as a basis to move forward with a better understanding of the systems under development.

In this report we will cover three different methods of extraction. We used Understand, srcML, and our own application – DependencyExtractor. We will see how they each work, what results are obtained and the process to get there.

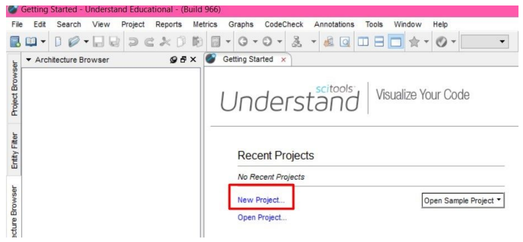
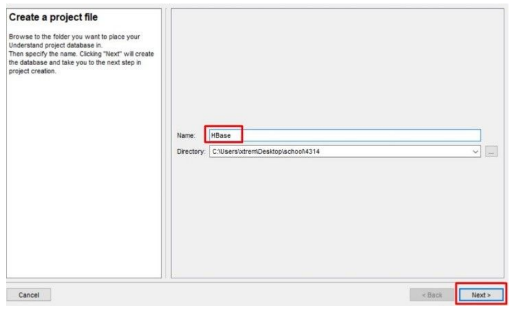
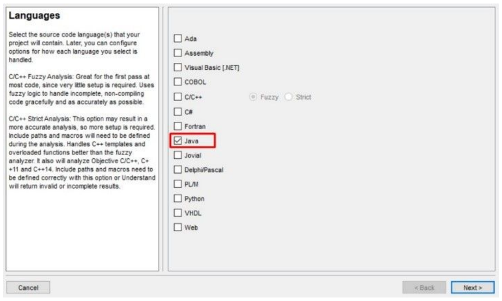
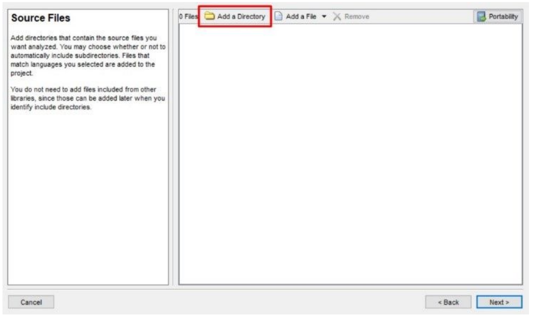
Extraction Technique: Scitools Understand

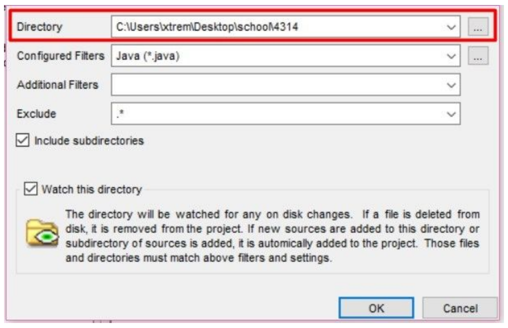
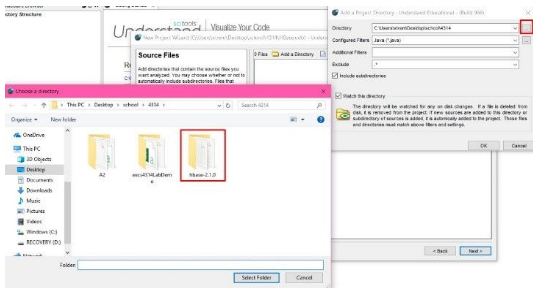
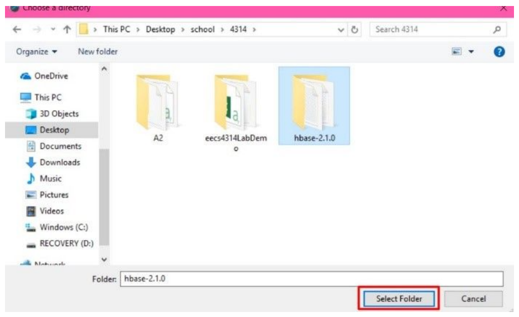
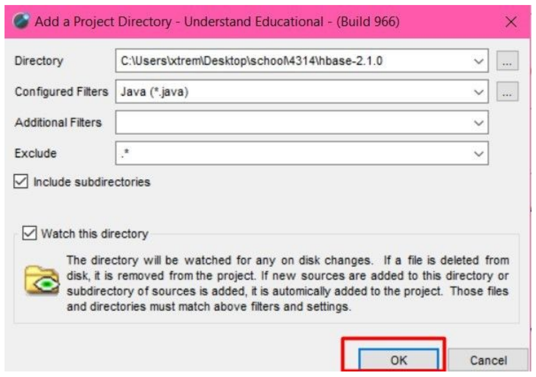
Understand is an IDE used by software developers in order to correctly and accurately document source code. It is an IDE that aids in the comprehension of legacy code as static code by analyzing it and visualizing and documenting it. [3] Understand does this by importing the source code files/directories and analyzing all the files. Understand supports a variety of languages such as C, C++, Visual Basic, Java, Assembly, PHP, JavaScript, XML, C#, Python, VHDL[4] and many more.

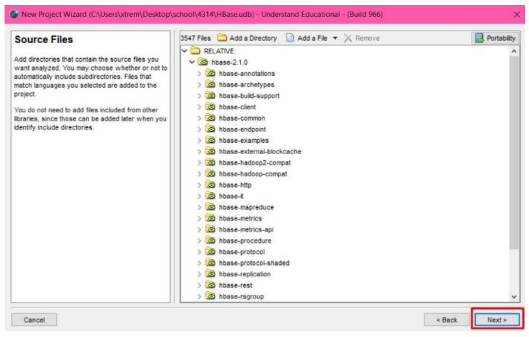
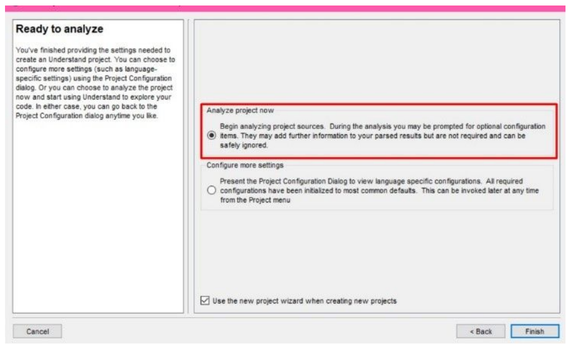
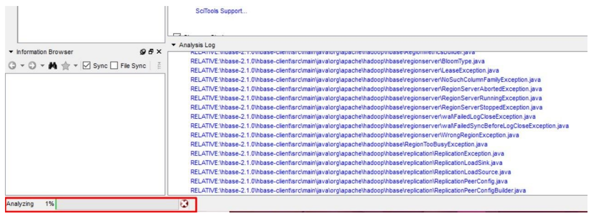
Understand has both advantages and disadvantages. As mentioned above, Understand supports a variety of languages for the source code. Once the code is imported and is analyzed, Understand then outputs warnings and errors where some 'vulnerabilities' are detected in the code with hyperlinks straight to the intended line of the intended file. These vulnerabilities are detected by incorporating a variety of security metrics in order to meet publishing standards for software such as "Effective C++", "MISRA-C 2004", "MISRA-C++ 2008"[5] and more. The user has the choice of whether to fix these errors or leave them be, but once that is done, a new developer can then decide to visualize the code based on dependencies via dependency graphs (between architectures, files or classes) or tree-maps (color-coded for readability).

Some issues, though, that come by when working with Understand are that it is a lot to take in if it's a first-time user. All the possibilities and what each of the features do and how to use them in order to get the most accurate result possible requires research and time in order to make full use of Understand. Another point is that, in the way that Understand detects vulnerabilities in the code, the set of rules it attempts to enforce on the code may not be complete. They will cover a huge majority of the vulnerabilities but there may also be many more of them that are not picked up by only using these rules. This may be an issue once the program is published and may not be detected beforehand. The final issue is that the source code's programming language(s) need to be known. Whether the information is given or the user has to delve into the directories in order to find what files are in the directories, the end result is a list of all the programming languages used in the project such that importing the code into Understand covers all the required files without missing any.

The task at hand was to extract the dependencies in HBase via Understand, in this case, and to do that, steps are shown in Table-1, on how the .csv file is extracted with the dependencies between them.

Step	Image	Description
1	 Getting Started - Understand Educational - (Build 966) Recent Projects *No Recent Projects* New Project... Open Project... Open Sample Project	First screen shown once *Understand* is downloaded and installed on user's device. Picking the 'New Project' option will begin the process of importing the source code into *Understand*.
2	 **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: HBase Directory: C:\Users\xtrem\Desktop\school\4314	Initially, *Understand* gives the project the name 'MyUnderstandProject' This can then be changed to any name that is relevant to the project to be made by filling in the entry next to "Name" as shown by the first red box.
3	 **Languages** Select the source code language(s) that your project will contain. Later, you can configure options for how each language you select is handled. C/C++ Fuzzy Analysis: Great for the first pass at most code, since very little setup is required. Uses fuzzy logic to handle incomplete, non-compiling code gracefully and as accurately as possible. C/C++ Strict Analysis: This option may result in a more accurate analysis, so more setup is required. Include paths and macros will need to be defined during the analysis. Handles C++ templates and overloaded functions better than the fuzzy analyzer. It also will analyze Objective C/C++, C++11 and C++14. Include paths and macros need to be defined correctly with this option or Understand will return invalid or incomplete results. ☐ Ada ☐ Assembly ☐ Visual Basic [.NET] ☐ COBOL ☐ C/C++ (Fuzzy / Strict) ☐ C# ☐ Fortran ☑ Java ☐ Jovial ☐ Delphi/Pascal ☐ PL/M ☐ Python ☐ VHDL ☐ Web	This step requires the user to know which programming languages are used in the source code in order to correctly import all the required files. Shown is a list of supported languages by *Understand*. 'Java' is selected, solely, as it is what HBase source code is written in.
4	 **Source Files** Add directories that contain the source files you want analyzed. You may choose whether or not to automatically include subdirectories. Files that match languages you selected are added to the project. You do not need to add files included from other libraries, since those can be added later when you identify include directories. 0 Files	*Understand* supports adding separate files or whole directories to the project, but since HBase is all collected under one directory, 'Add a Directory' is the chosen option, as is shown by the red square.

5	 Directory C:\Users\xtrem\Desktop\school\4314 Configured Filters Java (*.java) Additional Filters Exclude .* ☑ Include subdirectories ☑ Watch this directory The directory will be watched for any on disk changes. If a file is deleted from disk, it is removed from the project. If new sources are added to this directory or subdirectory of sources is added, it is automically added to the project. Those files and directories must match above filters and settings. 	'Directory' represents the path to the directory that is to be imported. 'Configured Files' shows the list of file extensions that were checked from step 3 above. 'Additional Filters' and 'Exclude' are options there in case the user has other filters they would like to use.
6	 Understand Visualize Your Code Source Files Add a Directory Choose a directory This PC > Desktop > school > 4314 A2 eecs4314LabDemo hbase-2.1.0 Select Folder Cancel OK < Back Next >	Clicking on the '…' button next to the 'Directory' option (as shown by the square on the top-right), results in the second window opening up. Navigate to and choose the sought-after directory before moving on to the next step.
7	 Choose a directory This PC > Desktop > school > 4314 Search 4314 Organize New folder OneDrive This PC 3D Objects Desktop Documents Downloads Music Pictures Videos Windows (C:) RECOVERY (D:) Network A2 eecs4314LabDemo hbase-2.1.0 Select Folder Cancel Folder: hbase-2.1.0	Once the correct directory is chosen, 'Select Folder' should be clicked next in order to proceed to the next step as shown.
8	 Add a Project Directory - Understand Educational - (Build 966) Directory C:\Users\xtrem\Desktop\school\4314\hbase-2.1.0 Configured Filters Java (*.java) Additional Filters Exclude .* ☑ Include subdirectories ☑ Watch this directory The directory will be watched for any on disk changes. If a file is deleted from disk, it is removed from the project. If new sources are added to this directory or subdirectory of sources is added, it is automically added to the project. Those files and directories must match above filters and settings. 	Once 'Select Folder' is selected, the extra window will close and the same window as step 5 should be updated, in the 'Directory' entry, with the new path to the required directory. Before proceeding, ensure that 'Include Subdirectories'[1] and 'Watch this directory'[2] are both checked.

9	 New Project Wizard (C:\Users\xtrem\Desktop\school\4314\HBase.udb) - Understand Educational - (Build 966) **Source Files** Add directories that contain the source files you want analyzed. You may choose whether or not to automatically include subdirectories. Files that match languages you selected are added to the project. You do not need to add files included from other libraries, since those can be added later when you identify include directories. 3547 Files · Add a Directory · Add a File · Remove · Portability RELATIVE: hbase-2.1.0 hbase-annotations hbase-archetypes hbase-build-support hbase-client hbase-common hbase-endpoint hbase-examples hbase-external-blockcache hbase-hadoop2-compat hbase-hadoop-compat hbase-http hbase-it hbase-mapreduce hbase-metrics hbase-metrics-api hbase-procedure hbase-protocol hbase-protocol-shaded hbase-replication hbase-rest hbase-rsgroup Cancel · < Back · Next >	Ensure that all the subdirectories of the required directory are visible in the 'Source Files' window, as shown. Once the directories are confirmed, move on to the next step by pressing on 'Next'.
10	 **Ready to analyze** You've finished providing the settings needed to create an Understand project. You can choose to configure more settings (such as language-specific settings) using the Project Configuration dialog. Or you can choose to analyze the project now and start using Understand to explore your code. In either case, you can go back to the Project Configuration dialog anytime you like. Analyze project now ● Begin analyzing project sources. During the analysis you may be prompted for optional configuration items. They may add further information to your parsed results but are not required and can be safely ignored. Configure more settings ○ Present the Project Configuration Dialog to view language specific configurations. All required configurations have been initialized to most common defaults. This can be invoked later at any time from the Project menu ☑ Use the new project wizard when creating new projects Cancel · < Back · Finish	This step is the last step through this project creating wizard, and this step allows the user to pick between configuring *more* settings, should they like by choosing the 'Configure more settings' option, or proceed through and finish the project creation by choosing the 'Analyze project now' option, as is shown, and then the 'Finish' button in order to finalize the project import.
11	 SciTools Support... Information Browser · ☑ Sync ☐ File Sync Analysis Log RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\regionserver\BloomType.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\regionserver\LeaseException.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\regionserver\NoSuchColumnFamilyException.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\regionserver\RegionServerAbortedException.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\regionserver\RegionServerRunningException.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\regionserver\RegionServerStoppedException.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\regionserver\wal\FailedLogCloseException.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\regionserver\wal\FailedSyncBeforeLogCloseException.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\regionserver\WrongRegionException.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\RegionTooBusyException.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\replication\ReplicationException.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\replication\ReplicationLoadSink.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\replication\ReplicationLoadSource.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\replication\ReplicationPeerConfig.java RELATIVE:\hbase-2.1.0\hbase-client\src\main\java\org\apache\hadoop\hbase\replication\ReplicationPeerConfigBuilder.java Analyzing 1%	Shown to the left is *Understand* analyzing through all the files in the imported directory for any 'vulnerabilities', as mentioned above.

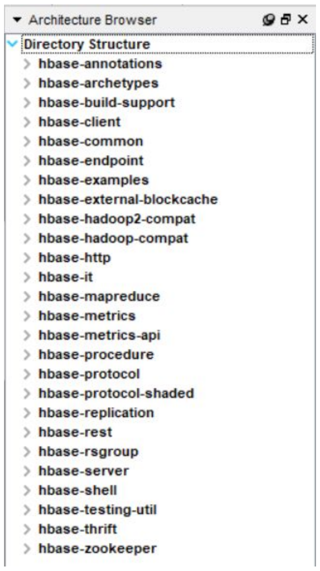
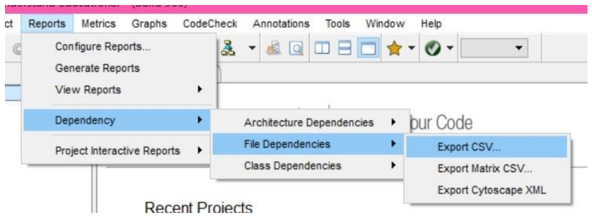
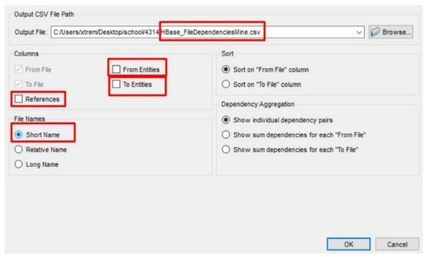
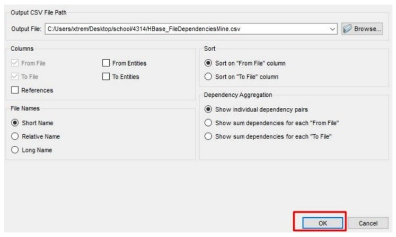
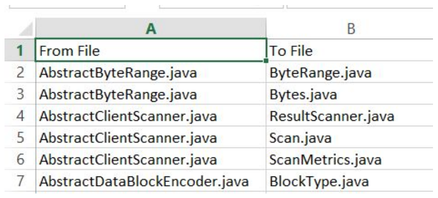
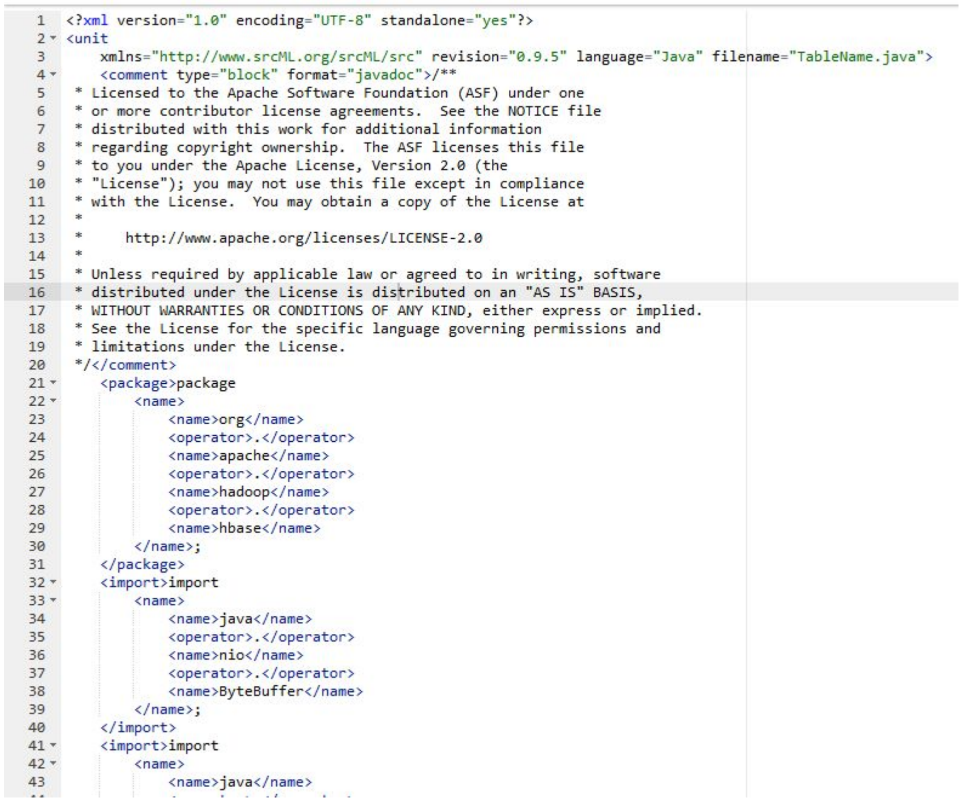
12	 Architecture Browser Directory Structure hbase-annotations hbase-archetypes hbase-build-support hbase-client hbase-common hbase-endpoint hbase-examples hbase-external-blockcache hbase-hadoop2-compat hbase-hadoop-compat hbase-http hbase-it hbase-mapreduce hbase-metrics hbase-metrics-api hbase-procedure hbase-protocol hbase-protocol-shaded hbase-replication hbase-rest hbase-rsgroup hbase-server hbase-shell hbase-testing-util hbase-thrift hbase-zookeeper	Once the analysis is over, under the 'Architecture Browser' tab, the user should see all the top-level subdirectories of the imported directory. This allows for quick and easy maneuverability through the files and/or directories for further research. *Understand* has already created a cache[6] at each file with a list of 'Depends on' and 'Depended on by' files for quick future use.
13	 ct Reports Metrics Graphs CodeCheck Annotations Tools Window Help Configure Reports... Generate Reports View Reports Dependency Project Interactive Reports Architecture Dependencies File Dependencies Class Dependencies Export CSV... Export Matrix CSV... Export Cytoscape XML our Code Recent Projects	To export the dependencies, clicking on 'Reports' then hovering over 'Dependency', 'File Dependencies' and then clicking on 'Export CSV' starts the wizard for dependency extraction.
14	 Output CSV File Path Output File: C:/Users/xtrem/Desktop/school/4314/HBase_FileDependenciesMine.csv Columns [✓] From File [✓] To File [ ] References [ ] From Entities [ ] To Entities File Names [✓ Short Name] Relative Name Long Name Sort [✓ Sort on "From File" column] Sort on "To File" column Dependency Aggregation [✓ Show individual dependency pairs] Show sum dependencies for each "From File" Show sum dependencies for each "To File" 	Rename the file to be exported by changing the keyword directly before the '.csv'. Next, the user can choose which options to use[3].
15	 Output CSV File Path Output File: C:/Users/xtrem/Desktop/school/4314/HBase_FileDependenciesMine.csv Columns [✓] From File [✓] To File [ ] References [ ] From Entities [ ] To Entities File Names [✓ Short Name] Relative Name Long Name Sort [✓ Sort on "From File" column] Sort on "To File" column Dependency Aggregation [✓ Show individual dependency pairs] Show sum dependencies for each "From File" Show sum dependencies for each "To File" 	Once the preferred options are (un)checked, finalize the exporting of the dependencies by pressing the 'OK' button as shown.
16	 | | A | B | | 1 | From File | To File | | 2 | AbstractByteRange.java | ByteRange.java | | 3 | AbstractByteRange.java | Bytes.java | | 4 | AbstractClientScanner.java | ResultScanner.java | | 5 | AbstractClientScanner.java | Scan.java | | 6 | AbstractClientScanner.java | ScanMetrics.java | | 7 | AbstractDataBlockEncoder.java | BlockType.java |	Shown is a sample of what the extracted dependencies looks like when the .csv file is opened with Microsoft Excel.

Table 1 – Steps to extract the dependencies into a .csv file using *Scitools: Understand*

Extraction Technique: srcML

srcML is a program that extracts information from source code provided to it. The information from the source code is made into an XML file which stores the information in a hierarchical structure. This XML file can then be searched for tags in the structure or keywords in the data to extract the desired information. srcML handles source code written in C, C++, C#, and Java[1]. HBase source code is written in Java.

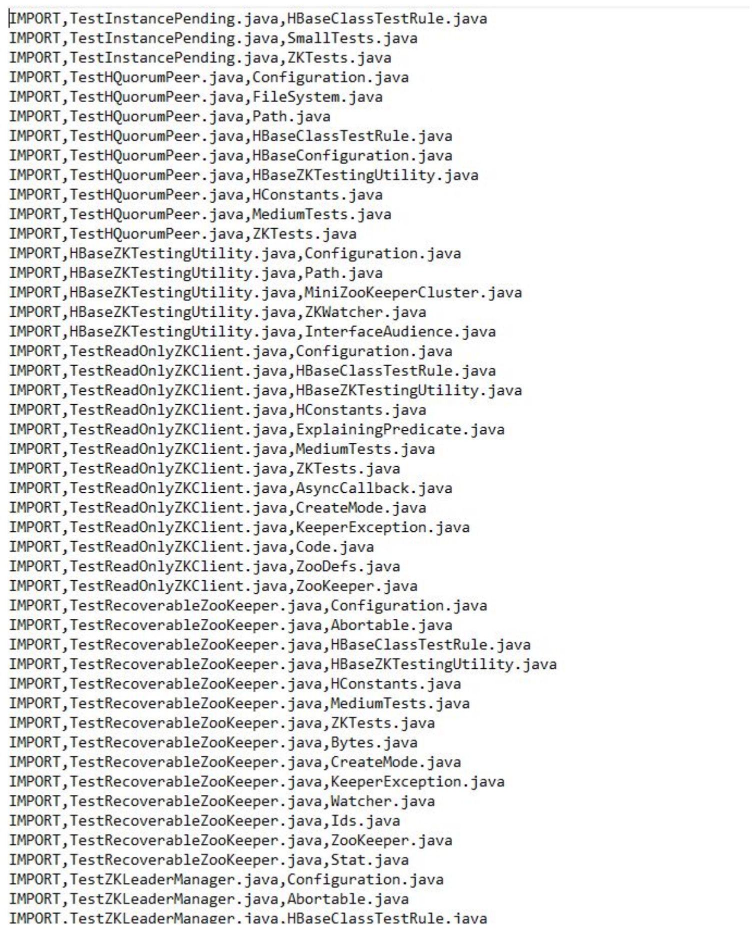


```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<unit
    xmlns="http://www.srcML.org/srcML/src" revision="0.9.5" language="Java" filename="TableName.java">
    <comment type="block" format="javadoc">/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */</comment>
    <package>package
        <name>
            <name>org</name>
            <operator>.</operator>
            <name>apache</name>
            <operator>.</operator>
            <name>hadoop</name>
            <operator>.</operator>
            <name>hbase</name>
        </name>;
    </package>
    <import>import
        <name>
            <name>java</name>
            <operator>.</operator>
            <name>nio</name>
            <operator>.</operator>
            <name>ByteBuffer</name>
        </name>;
    </import>
    <import>import
        <name>
            <name>java</name>
```

Figure 1 - Source code to XML Example

The first step to extracting the dependencies from the source code of HBase is to run it through srcML. This produces a large XML file, 150MB in size, that stores all of the classes of HBase. Figure 1, below, shows an example of the XML output. Note the hierarchy of tags, denoted with matching "<" and ">" characters, containing data between them.

After the XML file for the HBase code has been generated the next step is to extract the desired information. This is done by looking for relevant tags and taking the data contained within. To establish a dependency network the "import" tags were used. A short python script, Appendix C.1, finds import tags and creates an output file that lists all of the discovered import relationships. An example section of this list is shown in Figure 2. This parsing process created roughly twenty eight thousand relationships and cut the data file size from 150MB to 8MB.



```
IMPORT,TestInstancePending.java,HBaseClassTestRule.java
IMPORT,TestInstancePending.java,SmallTests.java
IMPORT,TestInstancePending.java,ZKTests.java
IMPORT,TestHQuorumPeer.java,Configuration.java
IMPORT,TestHQuorumPeer.java,FileSystem.java
IMPORT,TestHQuorumPeer.java,Path.java
IMPORT,TestHQuorumPeer.java,HBaseClassTestRule.java
IMPORT,TestHQuorumPeer.java,HBaseConfiguration.java
IMPORT,TestHQuorumPeer.java,HBaseZKTestingUtility.java
IMPORT,TestHQuorumPeer.java,HConstants.java
IMPORT,TestHQuorumPeer.java,MediumTests.java
IMPORT,TestHQuorumPeer.java,ZKTests.java
IMPORT,HBaseZKTestingUtility.java,Configuration.java
IMPORT,HBaseZKTestingUtility.java,Path.java
IMPORT,HBaseZKTestingUtility.java,MiniZooKeeperCluster.java
IMPORT,HBaseZKTestingUtility.java,ZKWatcher.java
IMPORT,HBaseZKTestingUtility.java,InterfaceAudience.java
IMPORT,TestReadOnlyZKClient.java,Configuration.java
IMPORT,TestReadOnlyZKClient.java,HBaseClassTestRule.java
IMPORT,TestReadOnlyZKClient.java,HBaseZKTestingUtility.java
IMPORT,TestReadOnlyZKClient.java,HConstants.java
IMPORT,TestReadOnlyZKClient.java,ExplainingPredicate.java
IMPORT,TestReadOnlyZKClient.java,MediumTests.java
IMPORT,TestReadOnlyZKClient.java,ZKTests.java
IMPORT,TestReadOnlyZKClient.java,AsyncCallback.java
IMPORT,TestReadOnlyZKClient.java,CreateMode.java
IMPORT,TestReadOnlyZKClient.java,KeeperException.java
IMPORT,TestReadOnlyZKClient.java,Code.java
IMPORT,TestReadOnlyZKClient.java,ZooDefs.java
IMPORT,TestReadOnlyZKClient.java,ZooKeeper.java
IMPORT,TestRecoverableZooKeeper.java,Configuration.java
IMPORT,TestRecoverableZooKeeper.java,Abortable.java
IMPORT,TestRecoverableZooKeeper.java,HBaseClassTestRule.java
IMPORT,TestRecoverableZooKeeper.java,HBaseZKTestingUtility.java
IMPORT,TestRecoverableZooKeeper.java,HConstants.java
IMPORT,TestRecoverableZooKeeper.java,MediumTests.java
IMPORT,TestRecoverableZooKeeper.java,ZKTests.java
IMPORT,TestRecoverableZooKeeper.java,Bytes.java
IMPORT,TestRecoverableZooKeeper.java,CreateMode.java
IMPORT,TestRecoverableZooKeeper.java,KeeperException.java
IMPORT,TestRecoverableZooKeeper.java,Watcher.java
IMPORT,TestRecoverableZooKeeper.java,Ids.java
IMPORT,TestRecoverableZooKeeper.java,ZooKeeper.java
IMPORT,TestRecoverableZooKeeper.java,Stat.java
IMPORT,TestZKLeaderManager.java,Configuration.java
IMPORT,TestZKLeaderManager.java,Abortable.java
IMPORT,TestZKLeaderManager.java,HBaseClassTestRule.java
```

Figure 2 - Example output from parsed XML file.

Extraction Technique: Import Statements

The third extraction technique that was used to explore HBase was a tool called DependencyExtractor. DependencyExtractor is a Java developed tool that sifts through the import statements in an applications source code for imports that match parameters supplied by the user (for example ‘HBase’ or ‘Apache’). In figure 3, it is shown how DependencyExtractor is built, in that it has 3 central systems. In the extractor package are contained all the classes that have a hand in direct import statement extraction logic. In the FileSystem class reside all the logic that has to do with importing files, breaking down files, and analyzing files at a shallow level.

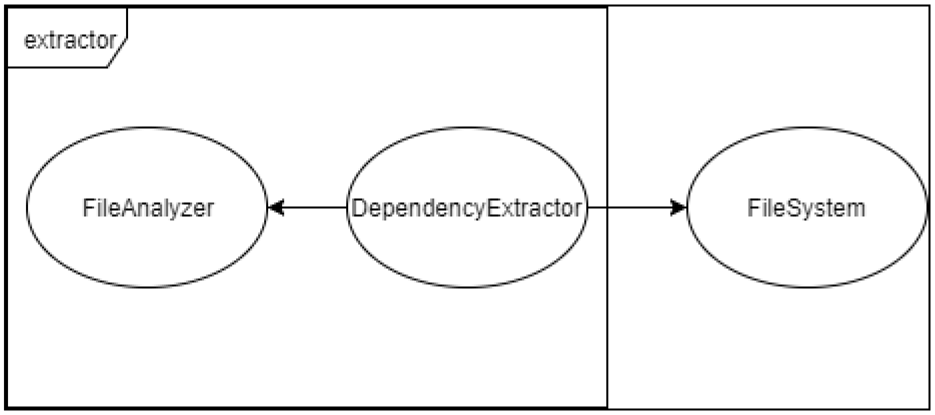


Figure 3 - DependencyExtractor Architecture

DependencyExtractor Usecase

In figure 4, a use case for DependencyExtractor is shown where it can be seen how it operates on an applications source code. First the root directory of the source code is provided to the top level DependencyExtractor class. This directory is then provided to FileSystem, where all the file names and locations are extracted and sent back over to DependencyExtractor. Using this information, FileAnalyzer then extracts all the import statements and deciphers the dependencies for each one. This list is then sent back to DependencyExtractor, which then sends the results back to the user.

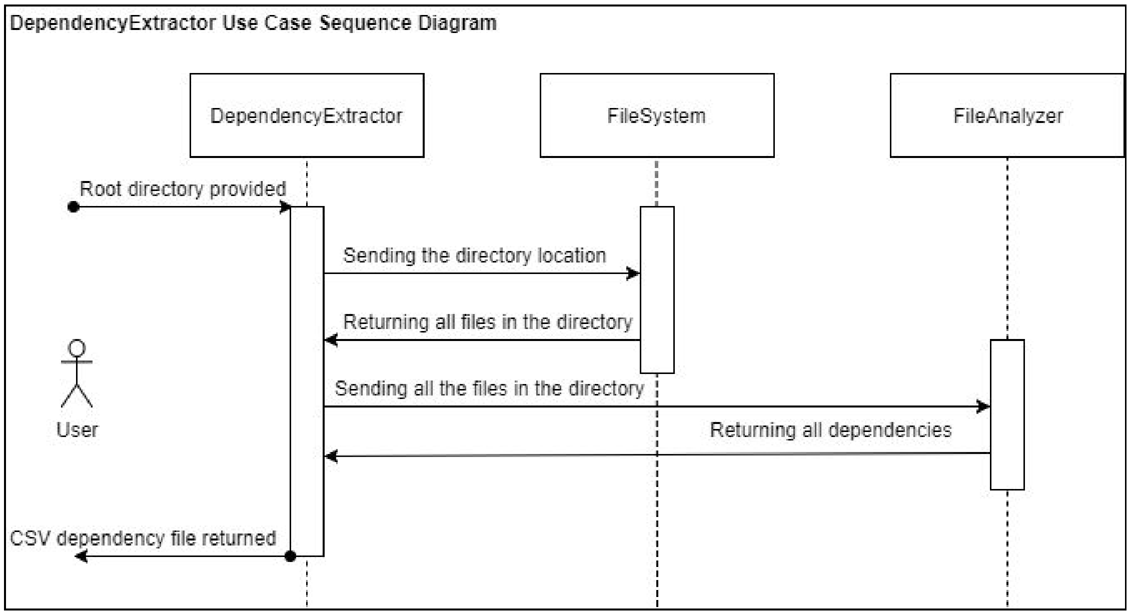

DependencyExtractor Use Case Sequence Diagram
DependencyExtractor
FileSystem
FileAnalyzer
Root directory provided
Sending the directory location
Returning all files in the directory
Sending all the files in the directory
Returning all dependencies
User
CSV dependency file returned

Figure 4 - DependencyExtractor Usecase

Stengths and Weaknesses of DependencyExtractor

The strengths of this method of dependency extraction is that it's virtually limitless in functionality and power. However, this comes at a major weakness of this method, and that is that everything must be done by hand and built up by the analyzer. This takes a large amount of time, and the amount of time and effort increases exponentially with the features that need to be added to the extractor. Ultimately it concludes to being a lot of reinvention of the wheel, and while it may produce excellent unique results, the effort is usually not worth it. Unless a certain feature which is unavailable anywhere else is required, this method will usually be the most time consuming, with the least justification behind it. Another large weakness with looking at import statements is that it only catches dependencies to classes in foreign packages, which means it misses a lot of dependencies in the local package it resides in, which is a huge hit to Recall (discussed later in Qualitative Analysis).

Quantitative Analysis

Having used the 3 different extraction methods, we are left with the results seen in Figure 5 and 6. We have 3 text files that contain the dependencies extracted by each method. To compare them, first we formatted the text files to be of the same form (file, dependent file), and then we wrote a simple python script, Appendix C.1, that would give us, as output, the total of extracted dependencies and the comparison result for each method. The comparison result being the number of common extracted dependencies between two methods and the number of unique extracted dependencies to each method.

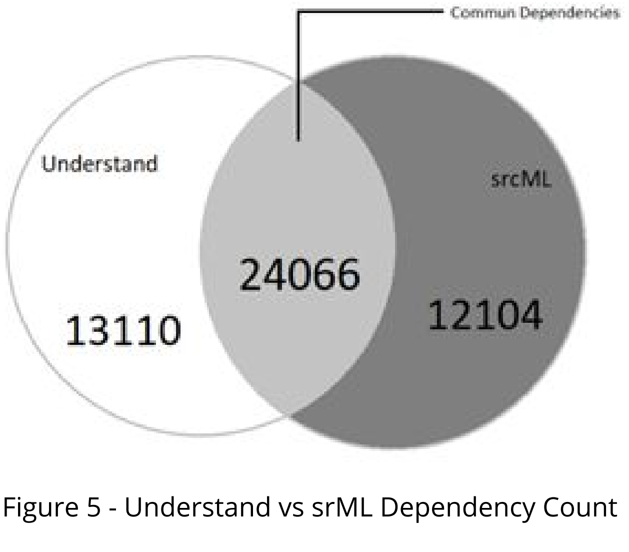


Figure 5 - Understand vs srML Dependency Count

We extracted 36170 unique dependencies using srcML, with the DependencyExtractor we got 28890 and finally with Understand we got 37176. Also, as we can see in Figure 5 when comparing Understands' results with srcMLs' results, they have 24066 common dependencies which means Understand has 35% unique dependencies and srcML has 33%. Looking at Figure 6, we can see, for the

DependencyExtractor and srcML that they have 28885 common dependencies which implies that srcML has 20% unique dependencies and the Dependency Extractor with only 5 unique dependencies, close to 0%. Finally, in the same Figure, we can see the results found for the last pair, Understand and the DependencyExtractor, are 23765 common dependencies which makes the DependencyExtractor with 18% unique dependencies and Understand with 36%.

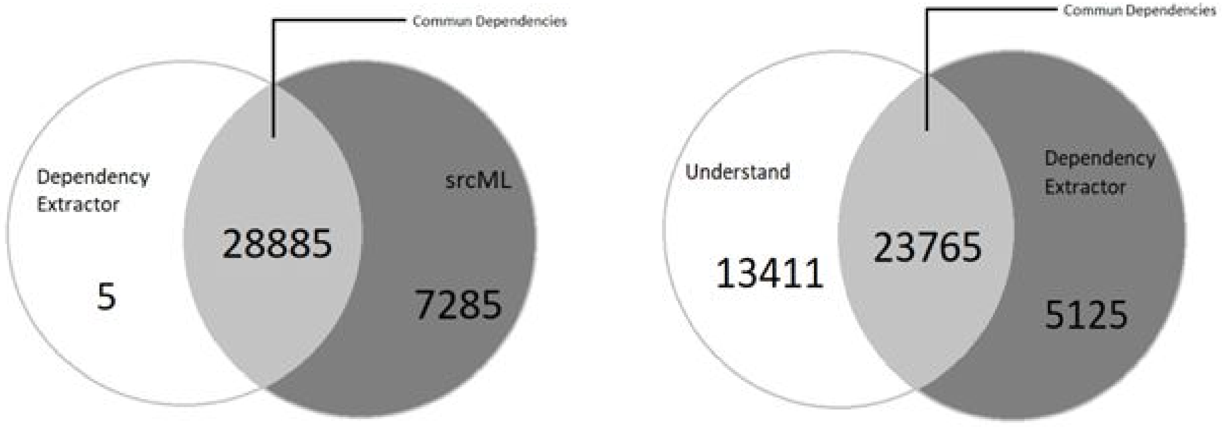
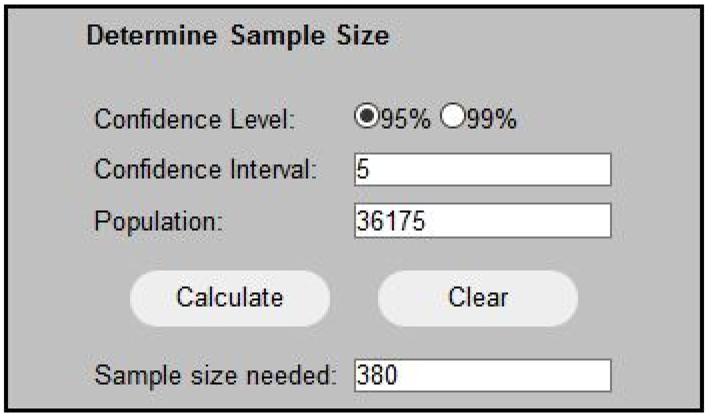


Figure 6 - DependencyExtractor vs srcML and Understand vs DependencyExtractor Dependency Count

From a first look perspective we could say that, from our results, the best extraction method would be Understand, since he has extracted the most dependencies and has the highest unique dependencies percentage in both comparisons (36% and 35%). The DependencyExtractor on the other hand didn’t fare so good, with the lowest amount of extracted dependencies and the lowest unique dependencies extracted percentage in both comparisons.

Qualitative Analysis

Before we begin qualitative analysis, a hypothesis will be made in order to make analysis simpler. The hypothesis is that our dependencies extracted using understand are all relevant, in other words, we would like to use our Understand dependencies as our oracle data. In order to legitimately do this, the Understand data will have to be validated, and in this way, our hypothesis will have to be validated. We will be validating our Understand data by sampling the dependencies according to a 95% confidence level with a 5% confidence interval. This gives us a sample size of 380 dependencies that must be validated (shown in figure 7, which is our sample size results). After randomizing our Understand data, we sample 380 random dependencies


Determine Sample Size

Confidence Level: [✓ 95%] 99%

Confidence Interval: 5

Population: 36175

Sample size needed: 380

Figure 7 - Sample Size Results Obtained Using surveysystem.com[Y]

After analyzing our sample size of 380 dependencies, we find that we have a relevance of ~98% or 372 relevant dependencies, and 8 dependencies that were irrelevant to our examination. An example of one of the irrelevant dependencies is AyncMetaTableAccessor.java -> Bytes.java. Bytes.java is a class in the util library of the apache package. When examined, it can clearly be seen that Bytes.java is not relevant to the inner workings of HBase and can easily be switched out with an equivalent Java library. After examining all the sample dependencies and their related source code, we are finally ready to move on to the next step. With a 98% relevancy, we can be assured that while not perfect, our Understand data is capable of being our oracle set.

srcML Precision and Recall

First, we will be analyzing the Precision and Recall of our srcML data against our Understand oracle. We will be using the data retrieved from the quantitative analysis section in order to complete our calculations for this analysis. This means that our total srcML dependencies are numbered at 36170, while our Understand dependencies number at 37176. The overlap between them is 24066 dependencies. The equations we will be using for Precision and Recall are as follows in figure 8.

$$\text{(1)}\quad \text{Precision} = \frac{\mathit{Relevant\ Dependencies} \cap \mathit{Retrieved\ Dependencies}}{\mathit{Retrieved\ Dependencies}}$$

$$\text{(2)}\quad \text{Recall} = \frac{\mathit{Relevant\ Dependencies} \cap \mathit{Retrieved\ Dependencies}}{\mathit{Relevant\ Dependencies}}$$

Figure 8 - Precision and Recall Equations

Thus, we can use these equations from figure 8 to aid in our calculations of srcML Precision and Recall. One thing to remember is that we will be multiplying our Precision and Recall by 98%, which is our confidence in our Oracle, in order to carry on the error across our calculations.

$$srcML\ Precision = \frac{Overlap}{srcML\ Dependencies} * 0.98 = \frac{24066}{36170} * 0.98 = 65.20\%$$

$$srcML\ Recall = \frac{Overlap}{Understand\ Dependencies} * 0.98 = \frac{24066}{36176} * 0.98 = 65.19\%$$

Figure 9 - Precision and Recall for srcML

From figure 9 it can be seen that our Precision and Recall for srcML are both around 65%, which is quite mediocre. It can be theorized then that due to the low precision, srcML seems to be using a shotgun approach for returning dependencies and thus giving us a large number of incorrect dependencies.

DependencyExtractor Precision and Recall

Using the same equations (1) and (2) for Precision and Recall from figure 9, we can then calculate our DependencyExtractor related values. We will again be using the data retrieved in the qualitative analysis for our calculations. Our Understand Dependencies number at 37176, while our DependencyExtractor dependencies number at 28890. The overlap between Understand and DependencyExtractor are 23765. Using these values, we can now calculate our Precision and Recall for DependencyExtrator.

$$DependencyExtractor\ Precision = \frac{Overlap}{DE\ Dependencies} * 0.98 = \frac{23765}{28890} * 0.98 = 80.62\%$$

$$DependencyExtractor\ Recall = \frac{Overlap}{Understand\ Dependencies} * 0.98 = \frac{23765}{36176} * 0.98 = 64.38\%$$

Figure 10 - Precision and Recall for DependencyExtractor

The high precision on our DependencyExtractor results in figure 10 imply that our extraction efforts from our import statements were successful and accurate. However, the low number of dependencies coupled with the low Recall imply that a lot of dependencies were left out. This is most likely due to the fact that only extracting from import statements misses out on all local package dependencies, and thus a lot of those dependencies went missing in the process, leaving us with a low total Recall.

Ultimately it can be seen from this excersize that DependencyExtractor is much more precise than srcML when it comes to extracting dependencies, however the low Recall for both of them imply very different things. For srcML, the low Recall reinforces the shotgun dependency extraction theory that we had previously, where a large number of total dependencies coupled with a low overlap implies a large number of false positives. However, in terms of our DependencyExtractor, it signals a massive flaw in our extraction using import statements, and that is that it leaves out all local package dependencies. This is one of the most likely reasons for the low overall dependencies coupled with high Precision. If DependencyExtractor was implemented further to extract local package dependencies, perhaps through method analysis, it could easily come to rival Understand, however this current flaw leaves it very weak in the meantime.

Limitations and Risks

While working on this paper we had to always be conscious about the methods used and the obtained results. That way we were forming strong opinions when using new tools and distinguishing better tools from worse tools. But we must always keep in mind that depending on the situation preferences might change. With that said, there are some key points worth mentioning in this section about our results and the methods used.

Looking at our DependencyExtractor, there are a number of issues with this method of extraction. First, writing code to extract dependencies is only as efficient as the time and effort spent on the code. While the flexibility and power of the end result were definitely boosted, much of it felt like reinventing the wheel, and it became a necessity to validate the results as they were untested. Another large problem is the inability to retrieve local package dependencies from import files, which led to a very low total dependency count. While Precision was high, Recall was low, and more dependencies from local packages would have definitely bumped up both numbers.

When using srcML we found ourselves with a very large XML file, very hard to read for a human being and were obligated to write a script to transform the data to readable text file. This limits srcML as an extraction method as it makes it seem as though work is being done twice over. Another large deficit of srcML is the fact that it extracted a very large amount of irrelevant dependencies, leading to a high dependency count, with a very low Precision.

For Understand we needed a properly configured and ordered source code directory for the representation to be accurate. Furthermore, the more about the code we knew the better we could configure Understand, making it a prerequisite to learn about the software you are trying to extract dependencies from. Not a very large problem, but definitely impeded quick and accurate extraction. However, Understand definitely produced the best results after configuration.

Lessons Learned

We learned through srcML that in order for us to use the software properly we needed to also work on translating the results as well as being able to use XPath for parsing. This is definitely a learning point as our previous experience with Understand was much more straightforward as everything was easily done and accessible via drop down menus. The major point we learned is that all 3 software had different prerequisites in order to get the extracted dependencies. However, our conclusion is that a deeper understanding of the source code leads to better results from Understand. Furthermore, a lot of effort and time spent on the import extractor led to respectable results, at the cost of reinventing the wheel somewhat. Ultimately though, srcML had major shortcomings with its specific XPath language and need to retranslate results to be human readable.

Conclusion

After using the three different methods to extract file dependencies and performing qualitative and quantitative analyses on the results, we found that a lot of the tools had a huge learning curve in order to output proper data. Furthermore, the data was not always completely accurate or relevant, leading to confusion at times. Developers and engineers have seen the need and importance of these tools to maintain and create software, so it is likely better tools will become available in the future. However for the moment, it seems that nothing replaces experience and dedication when it comes to architecture extraction.

Appendix A: Naming Conventions

Abbreviation	Explanation
XML	e**X**tensible **M**arkup **L**anguage - A programming markup language designed for storing and transporting data. Associates data with a hierarchical structure and uses tags to describe different types of data that may be described.
MB	**M**ega**B**yte - 2^{20} bytes of digital information.
CSV	**C**omma **S**eparated **V**alues - A style of information formatting commonly used for data intended to be entered into spreadsheets. Rows of data are separated by line, columns are separated by commas.
IDE	**I**ntegrated **D**evelopment **E**nvironment - Any program that combines tools and features related to writing or testing code.

Appendix B: Bibliography

[1] “SrcML Beta v0.9.5.” *SrcML*, www.srcml.org/.

[2] *An Introduction to Biometric Recognition - IEEE Journals & Magazine*, Wiley-IEEE Press, ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1311060.

[3] “Understand (Software).” *Wikipedia*, Wikimedia Foundation, 17 Oct. 2018, en.wikipedia.org/wiki/Understand_(software).

[4] “Supported Languages.” *Visualize Your Code*, scitools.com/feature/supported-languages/.

[5] “Published Coding Standards.” *Visualize Your Code*, scitools.com/feature/published-coding-standards/.

[6] “Scitools Labs: Dependency Analysis.” *Visualize Your Code*, scitools.com/scitools-labs-dependency-analysis/.

Appendix C: Source Code

C.1 - srcML Parsing Script

```
import lxml
from lxml import etree


#Takes an array of elements and checks to see if any of
#those elements suggest their parent is part of hbase.
def is_in_hbase(child_array):
 for child in child_array:
  if child.text:
   if "apache" in child.text or "hadoop" in child.text or "hbase" in child.text:
    return True
 return False


tree = etree.parse("hbase_srcml_output.xml")
root = tree.getroot()
for unit in root:
 if "unit" in unit.tag:
  file_name = unit.get("filename")
  file_name = file_name.split("\\")[-1]

  for child in unit:
   if "import" in child.tag:
    for subchild in child:
     if is_in_hbase(subchild[:]):
      dependency_name = subchild[-1].text + ".java"
      print("IMPORT," + file_name + "," + dependency_name)
```

C.2 - Comparison script

```
data_a = set(open("A_dependency_extractor.txt","r").readlines())
data_b = set(open("B_srcml_dependencies.txt","r").readlines())
data_c = set(open("C_understand_dependencies.csv","r").readlines())

total_nr_dep_a = len(data_a)
total_nr_dep_b = len(data_b)
total_nr_dep_c = len(data_c)

print("The total number of dependencies for data_A is: " + str(total_nr_dep_a))
print("The total number of dependencies for data_B is: " + str(total_nr_dep_b))
print("The total number of dependencies for data_C is: " + str(total_nr_dep_c))

same_count = 0
missing_count = 0
for line in data_a:
if line in data_b:
       same_count += 1
else:
       missing_count += 1
```

```
print("Data set A and B share " + str(same_count) + " dependencies. Data set A has " +
str(missing_count) + " dependencies not mentioned in Data set B.")

same_count = 1
missing_count = 0
for line in data_b:
if line in data_a:
       same_count += 1
else:
       missing_count += 1

print("Data set B and A share " + str(same_count) + " dependencies. Data set B has " +
str(missing_count) + " dependencies not mentioned in Data set A.")


same_count = 0
missing_count = 0
for line in data_a:
if line in data_c:
       same_count += 1
else:
       missing_count += 1

print("Data set A and C share " + str(same_count) + " dependencies. Data set A has " +
str(missing_count) + " dependencies not mentioned in Data set C.")

same_count = 0
missing_count = 0
for line in data_c:
if line in data_a:
       same_count += 1
else:
       missing_count += 1

print("Data set C and A share " + str(same_count) + " dependencies. Data set C has " +
str(missing_count) + " dependencies not mentioned in Data set A.")

same_count = 0
missing_count = 0
for line in data_b:
if line in data_c:
       same_count += 1
else:
       missing_count += 1

print("Data set B and C share " + str(same_count) + " dependencies. Data set B has " +
str(missing_count) + " dependencies not mentioned in Data set C.")

same_count = 0
missing_count = 0
for line in data_c:
if line in data_b:
       same_count += 1
else:
       missing_count += 1

print("Data set C and B share " + str(same_count) + " dependencies. Data set C has " +
str(missing_count) + " dependencies not mentioned in Data set B.")
```