2016

Dependency Extraction

4314 ASSIGNMENT 3

REVERSE ‘EM ALL

Contents

Abstract

This report will begin by introducing three dependency extraction tools. These tools are Understand, Intellij IDEA, and a third program called Include, written by Reverse 'em All. We will discuss how they extract their dependencies. We will make two comparisons, one with Include vs. Understand and another with IDEA vs. Understand in the quantitative analysis. We'll discuss how many dependencies were extracted from each tool. In the qualitative analysis, we'll discuss our significant findings to fully comprehend what makes each extraction tool different. We'll discuss a fourth alternative tool called srcML, its strengths, and why we ultimately chose IDEA over srcML. Finally, we'll discuss about lessons we have learned during the creation of this report.

Introduction

With such a large-scale project like Apache Hadoop, there are many interactions between project files. Some files may call other files through methods, imports, object initializations and more. To understand how a large project works in practice, analyzing the dependencies between system files can give an overall view of how the project works. This report, using Apache Hadoop as an example, will consider several dependencies within Hadoop, and the reasoning behind these dependencies.

Although there are many dependency extraction tools available on the internet, this report will focus on three main ones. These tools are Understand, Intellij IDEA, and a third program called Include, written by Reverse 'em All. Since these programs were created by different groups of people, and they extract dependencies differently from each other, the number of dependencies they find may differ from one to another. How many dependencies they produce and the amount that are different from another will be analyzed. This report will focus on the dependencies of Understand vs. Include and Understand vs. Intellij IDEA. In Hadoop, the total number of dependencies extracted by these tools can reach into the ten thousand. There are many different types of programs written in different programming languages, so this report specifically focuses only of Java files

The inner coding that makes up Understand and IDEA are not well known, as such, we shall focus on what exactly makes them different. A qualitative analysis of the overlapping and unique dependencies between these extraction tools may give insight on how each tool determines whether one file depends on another, and the reasoning behind their dependency.

Finally, this report will briefly examine another extraction tool called srcML. Although we could not go into depth about the number of dependencies it can potentially extract like in the 3 main extraction tools, we'll explain how powerful srcML can be at being an accurate extraction tool. Figure 1 highlights the process which we extract dependencies from Hadoop and analyze their dependencies.

Dependency Extraction Process

- First phase extracting dependencies into outputs generated by the chosen tools.
- The comparison of dependency will be executed next, whereby for each pair of reports, we group the records by (1) overlapping results, (2) abstracted by Understand complement of the compared tools (3) abstracted by the compared tool complement of Understand.
- After deciding the confidence level and interval, and getting the number dependencies, we use an online calculator to determine the number of classes we must examine to get significant results. Then we divulge into studying the source code, and the depedency chosen is randomly picked.
- We finally do a qualitative analysis and attempt to explain our findings

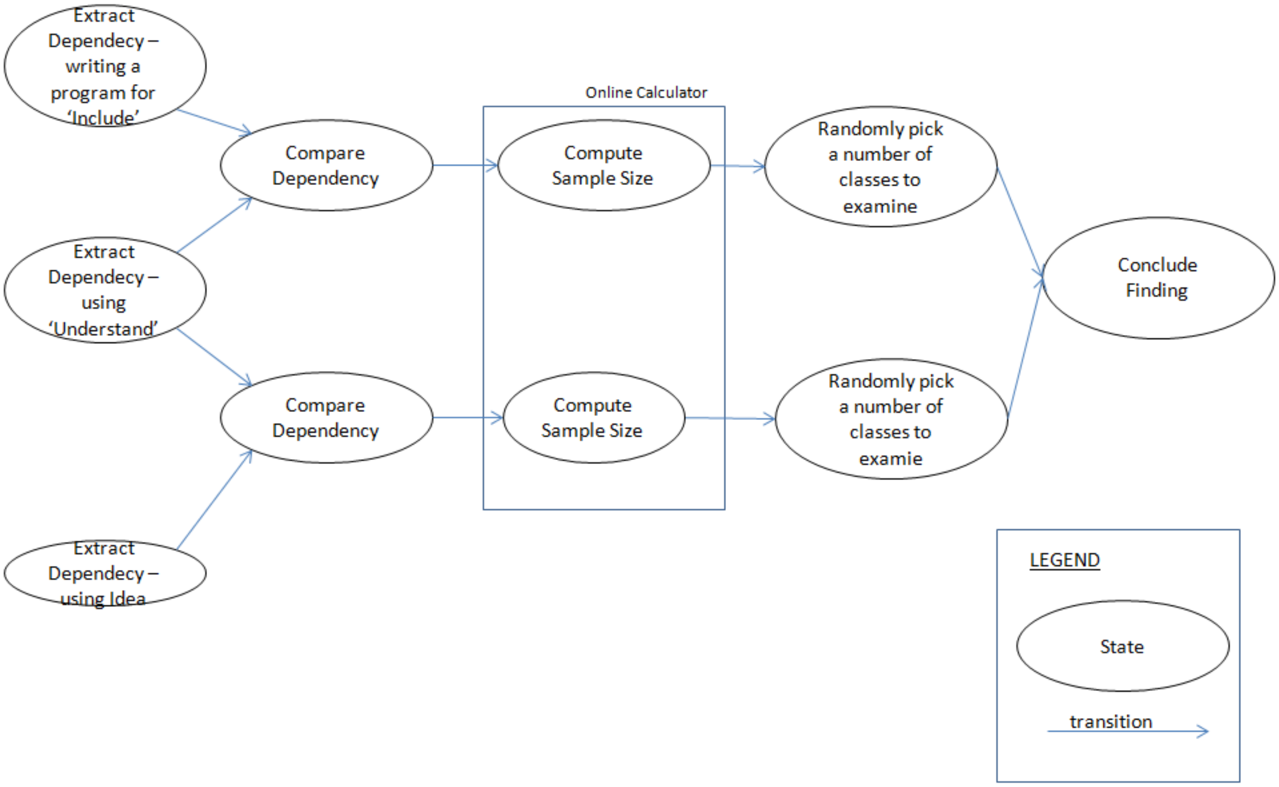

Extract Dependecy – writing a program for 'Include'
Extract Dependecy – using 'Understand'
Extract Dependecy – using Idea
Compare Dependency
Compare Dependency
Online Calculator
Compute Sample Size
Compute Sample Size
Randomly pick a number of classes to examine
Randomly pick a number of classes to examie
Conclude Finding
LEGEND
State
transition

Figure 1: State Diagram of the Extraction & Dependency Analysis Process

Dependency Extraction Tools

Understand

Understand is a static analysis tool focused on source code comprehension, metrics, and standards testing [1]. It can help a user understand exactly what a project is doing by analyzing data. It can measure, visualize, and maintain a project [1]. This report focused only on the dependencies Understand produces.

We used Understand to only focus on java files within the project. Understand can determine if one file depends on another by:

- Calls
- Includes/Imports
- Inheritance
- Object initializations
- Implementations
- Overrides
- Throws
- Uses
- Sets

By using the dependency browser, Understand highlights the specific dependency occurrence in the code. For example, if a method called `functionA` calls another method `(functionB)` from another class, clicking on the dependency line "*File_A.functionA Calls File_B.functionB at File_A*" shows exactly where in the code this dependency relation occurred. One can also open File_B and see which method was called. Both backwards and forward dependencies are featured in Understand, but we shall only focus on forward dependencies.

Understand exports Hadoop dependencies into a file of type .CSV. Each line is formatted as below. The first line is the path of the first file in the Haddop project, and the second line after the comma is the file that the first file depends on. The three numbers at the end are "reference", "from entities", and "to entities" respectively.

```
hadoop-2.7.3-src\hadoop-yarn-project\...
\registry\AbstractRegistryTest.java,hadoop-2.7.3-src\hadoop-common-
project\...\PathNotFoundException.java,2,2,1
```

Include

Include is a basic program produced in Java that looks into the import lines found in Hadoop files. Include can only extract dependencies from java files. Prior to running this program, there is another text file that has the directory pathways to each java file in the Hadoop system. The program can be explained through these steps:

1. It recursively searches through the Hadoop project directory and scans each entity in the file. If the entity is a directory, the function calls itself again to scan all entities in that directory. If the entity is a java file, then it continues.
2. It scans the java file for import statements, for example, "import (FILL IN)".
3. Once an import line is found, it scans the text file to see if the pathway exists.
4. If it exists, then this file depends on the import, and the import is a file in the system. It exports this dependency to an output text file.

The drawbacks for using this code is that it only considers java files that have an import statement. A file can depend on another through Java annotations. If the files have the same parent and are in the same directory or package, one file can call methods from another file without importing. Another drawback in using Include is it does not consider import statement wildcards, especially those that can import all files from a directory.

Fortunately, for the java files seen thus far up the to the making of this report, import statements explicitly state which file in a package it is importing. Java import libraries such as

```
import java.io.*;
```

are also ignored, because although the program does detect the import statement, when it scans the text file consisting of java file pathways, it will not be found since the java library is in within the Hadoop project.

Intellij IDEA

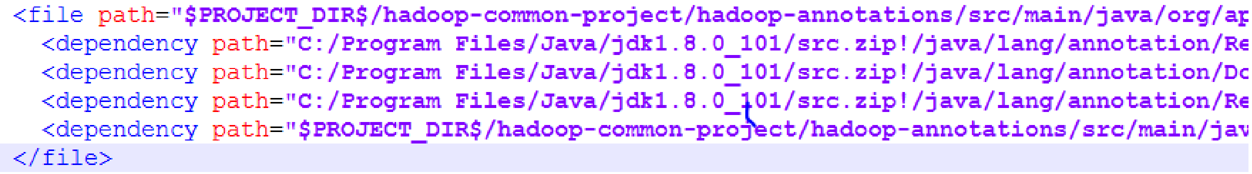
Initially we attempted to use srcML to extract dependency fact as is suggested. We gave up halfway through for the reasons given below,

1. Unlike srcML that we need to specify which type of information we want to extract (from tags <import>, <extend>, <implement>), IDEA provides a complete extraction scheme generating dependency fact by clicking a few buttons.
2. The output of IDEA in XML format is easier to process compared to the output of srcML.

Eventually, we determined to use a new tool called Intellij IDEA.

Extraction process

1. Download Intellij IDEA from its official website and install it.
2. Open Intellij IDEA from desktop. It is originally a Java IDE but it also has powerful module to extract dependency information.
3. Ensure the Hadoop project is correctly imported to workspace scope.
4. Select “Analyze” from menu bar then select “Analyze dependencies…”
5. When the result is out, click to export the result to XML form. The XML file looks like this,



```
<file path="$PROJECT_DIR$/hadoop-common-project/hadoop-annotations/src/main/java/org/ap
  <dependency path="C:/Program Files/Java/jdk1.8.0_101/src.zip!/java/lang/annotation/Re
  <dependency path="C:/Program Files/Java/jdk1.8.0_101/src.zip!/java/lang/annotation/Do
  <dependency path="C:/Program Files/Java/jdk1.8.0_101/src.zip!/java/lang/annotation/Re
  <dependency path="$PROJECT_DIR$/hadoop-common-project/hadoop-annotations/src/main/jav
</file>
```

6. We do some simple text processing using Notepad plug-ins (e.g. replace relative path for absolute part; remove all Java JDK dependencies.) before applying the XML parsing program upon the XML file.

7. Apply the XML program upon the XML file to output a plain text file which can be used as input for the Comparison.java program.
8. We found that some .xml exists in dependency relations. We split the output plain text to two sets of dependency relations (java dependency only or contains non-java dependency) which may be used in further research.

Quantitative Analysis of Include vs. Understand

Comparison.java

We built another program that take the dependency file from Understand and Include/IDEA, and outputs a text file containing which dependencies occurred in both extraction tools, dependencies occurring only in Understand, and dependencies occurring only in Include. It simply modifies the files from both so that a direct comparison can made.

In the text file exported by Comparison.java, the number of dependencies in Hadoop reached in the tens of thousands.

Table 1: Table of the number of dependencies in Understand and Include contains the amount of dependencies in Include and Understand, as well as the overlapping dependencies. Figure 2 shows the percentage of dependencies that overlap, and the percentage exclusive to each extraction tool. Most dependencies are captured by both Include and Understand.

Table 1: Table of the number of dependencies in Understand and Include

Understand - Include	Include ∩ Understand	Include - Understand	Total
21830	40404	889	**63123**
34.583%	64.008%	1.408%	100.000%

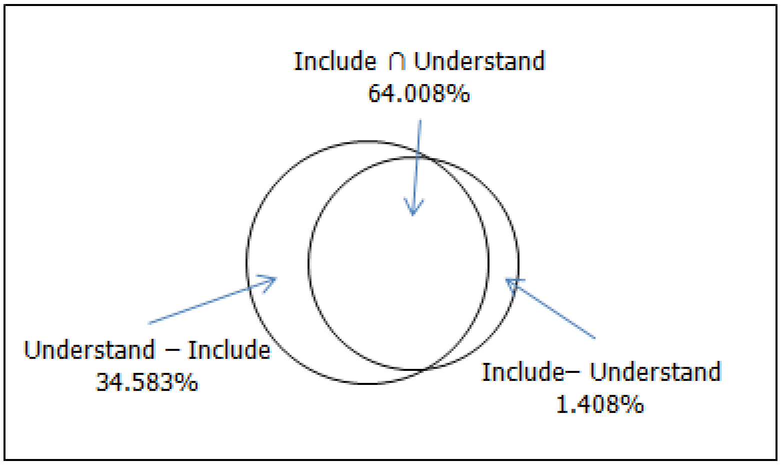


Figure 2: Venn diagram of the comparison between Include and Understand

A total of 63123 relations were extracted. **Error! Reference source not found.** and **Error! Reference source not found.** are pie charts of Understand vs. Include, and Include vs. Understand, respectively. In **Error! Reference source not found.**, only 1% of dependencies are exclusive to Include. The meaning behind these numbers will be explained in the quantitative analysis.

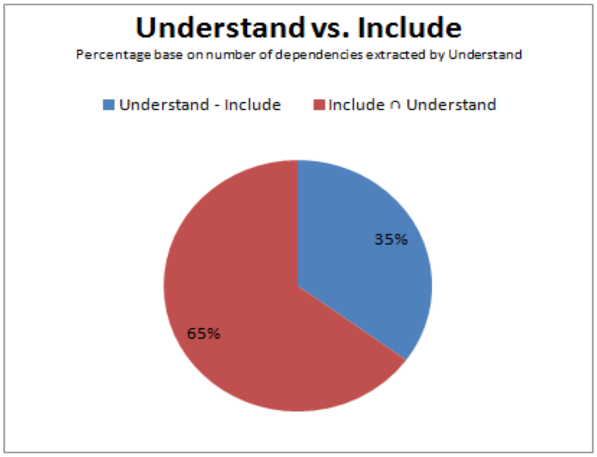


Figure 3: Pie chart of the dependences in Understand vs. Include

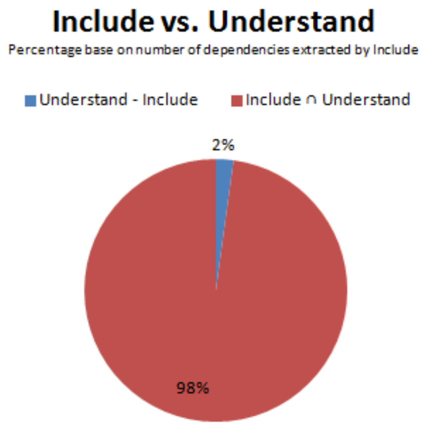


Figure 4: Pie chart of the dependences in Include vs. Understand

Sample size

Since it would take a long time to examine 60, 000 dependencies, we obtained a sample size from the total amount of dependencies. We used a confidence level of 95% and a confidence interval of 5% to obtain a sample size of 382 for Include vs. Understand. This allows us to give the most accurate depiction of the population. We then used a random number generator and selected 382 numbers from a range of 1 to 63, 123. The random numbers refer to the dependency in that population that we examined.

Include vs. Understand Qualitative:

With a sample size of 382 over the total population of Understand (35%), Include (1%) and the overlap of the two (64%) it was prudent to determine the unique differences to each dependency. Looking at 133 unique samples from Understand, 244 samples from their overlap of dependencies, but only taking 1% of samples for Include would not be sufficient to find the reason for distinct dependencies thus we will take 10% of total samples and delegate them from overlap samples making Include samples 38 and 206 overlap samples. This will give a good idea of the quality that each tool has and determine which is the better tool to use. We will now look at some examples of the majority results found from the samples above. The blue file path will represent the dependent and the red file path will represent the depended.

Overlap

Example (1): …\hadoop\hdfs\server\namenode\FSDirSymlinkOp.java →
…\hadoop\fs\permission\**PermissionStatus**.java

In this dependency, it was found that the dependent imports the depended and that the depended is an abstract data type PermissionStatus and is used as so in the dependent:

```
PermissionSatus dirPerms;
        …
dirPerms.getUserName;
```

Thus, here we see that both Include and Understand have found import statements, which that is what Include only does, but Understand also finds method parameters and method calls. This is just one example of the samples that we took here are just a few more that exhibit the same or similar dependency qualities found.

Example (2): …\hadoop\fs\AbstractFileSystem.java →
…\hadoop\util\Progressable.java

It was found in fact that all samples show these same qualities; an import of the depended is found in the dependent file which shows both Understand and Include both find import statements. Understand found it as a dependency as well because there was a use of the depended as an abstract data type, method calls, variable use, an interface for another class, inheritance, implementations and others that the Understand tool seeks for as described earlier in the paper. Now we will look at some examples that only the Understand tool found.

Understand Only

Example (1): ...\java\org\apache\hadoop\security\TestNetgroupCache.java →
…\main\java\org\apache\hadoop\security\NetgroupCache.java

Here we have a unique to Understand dependency found from the samples. There is no import statement which we know is the only way that Include would find a dependency. Understand finds this dependency because the dependent uses a method from the depended. The reason there is no import for this particular dependency is because the files are in the same directory so the import is not needed.

Example (2): …\hadoop-common\src\test\java\org\apache\hadoop\crypto\TestCryptoCodec.java →
…\hadoop-common\src\main\java\org\apache\hadoop\crypto\OpensslCipher.java

In this case, again there is no import of the depended which clarifies why the Include tool did not find the dependency. The reason the Understand tool found it was because the dependent uses the test package that the depended is in but not directly, the dependent also uses a child of the depended' method thus there is a dependency in two more ways that Include would not find, imports of packages as they don't directly include the filename of the depended and the use of a child's method.

All the dependencies exclusive to Understand don't include an import but have a dependency that Understand finds through either the import of a package that the dependent belongs too, an import and use of a child of the depended, or singularly or a combination of the former but the depended belongs to the same package/directory as the dependent allowing use without an import. Next we will look at some examples of dependencies only found by the Include tool.

Include Only

Example (1): EEcasd…\hadoop\yarn\server\nodemanager\containermanager\container\ContainerImpl.java →
…\hadoop\yarn\server\nodemanager\containermanager\application\
ApplicationContainerFinishedEvent.java

This is a case where only the Include found a dependency. The depended is imported which is how Include finds dependencies, but Understand also find these. An interesting thing to note is that although this dependency was not outputted to the csv file it is in the Understand Dependency Browser, curiously the depended is used as so:

```
…
eventHandler.handle(new ApplicationContainerFinishedEvent(containerId))
…
```

It directly uses a constructor of the class as a parameter. It is possible that Understand does not find these unique dependencies, although finding it in the Understand Dependency Browser contradicts this statement indicating an error and affecting the precision of the Understand tool. Here are a couple other examples of the same situation described:

Example (2): …\hadoop\yarn\server\applicationhistoryservice\ApplicationHistoryWriter.java →
…\hadoop\yarn\server\applicationhistoryservice\records\
ApplicationAttemptFinishData.java

We should note that all the samples explored exhibited the situation described earlier so quantitatively we can estimate that all the 889 unique dependencies found by Include are either an error on our part of the use of Understand or an error in the precision of the tool. Now we will see the qualitative precision and recall of both tools against each other:

Include - Precision = 40404/41293 ~ 0.98
Recall = 40404/63123 ~ 0.64
*Note that recall would normally use only Include acquired samples

Understand - Precision = 62234/63123 ~ 0.99

*Note that recall cannot be considered fully as errors cannot by guaranteed and this precision is based on comparison population and not acquired by Understand only population

Quantitative Analysis of Understand Versus Intellij IDEA

Table 2: Table of the number of dependencies in Understand and Intellij IDEA

Understand - Intellij	Intellij ∩ Understand	Intellij - Understand	TOTAL
11,375	50,859	3,416	65,650
17.327%	77.470%	5.203%	100%

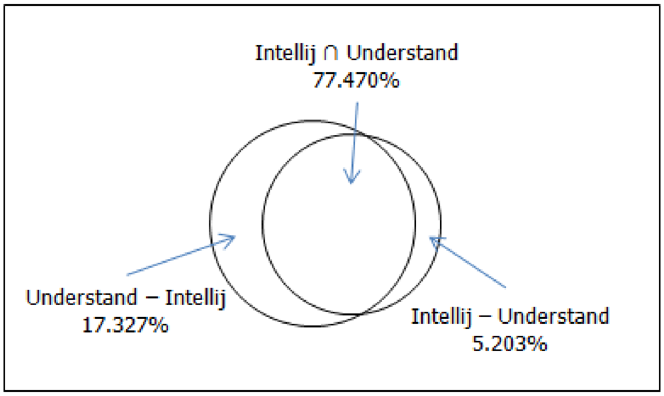


Figure 5: Venn diagram of the comparison between Intellij and Understand

Intellij IDEA and Understand extracted a total of 65,650 dependencies, with 77.47% of the dependencies are captured by both tools. This percentage is smaller than that between Include and Understand. This indicate Intellij IDEA captures certain aspect that Understand ignores.

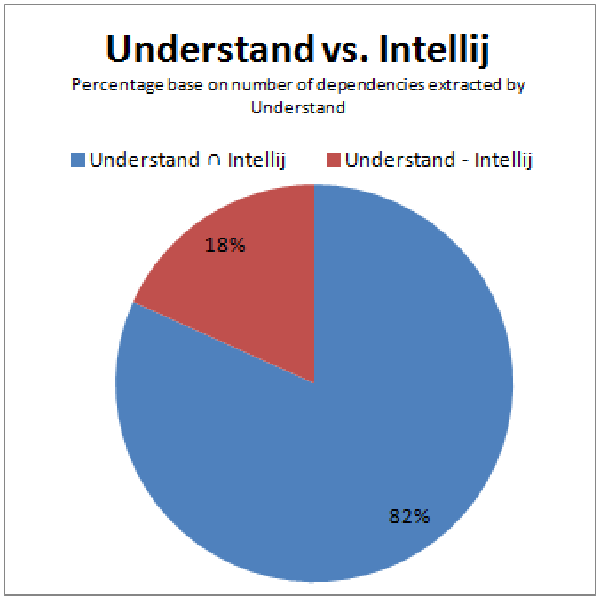


Figure 6: Pie chart of the dependences in Understand vs. Intellij

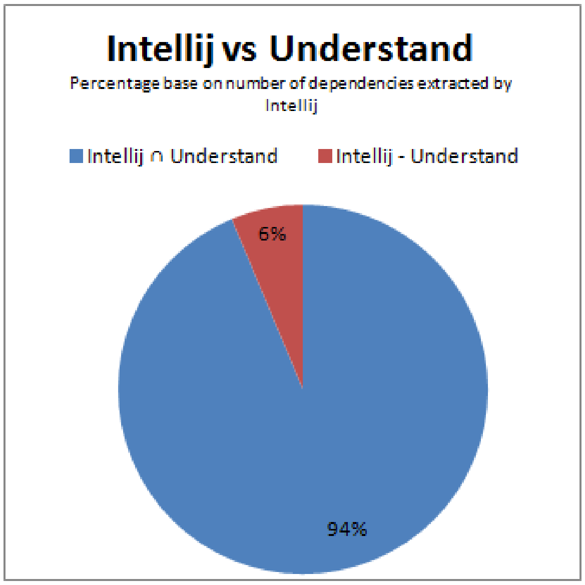


Figure 7: Pie chart of the dependencies in Intellij vs. Understand

18% of the dependencies that Understand extracted are not captured by Intellij, while 6% of the dependencies that Intellij extracted are not captured by Understand. Understand captures a substantial number of dependencies of the subsystem.

The precision of the IDEA tool is 50538/53954 ~ 0.94

The recall of the IDEA tool is 50538/65650 ~ 0.77

IDEA vs. Understand Qualitative

With a sample size of 382 over the total population of Understand only (17%), IDEA only (5%) and the overlap of the two (78%) it was prudent to determine the unique differences to each dependency. We will allocate samples to each category (Understand only, IDEA only, overlap) as per their share in the total population. So, we will have 65, 19, 298 samples from Understand only, IDEA only and overlap respectively. We will now look at some examples of the majority results found from the samples above. The blue file path will represent the dependent and the red file path will represent the depended.

Overlap

The below dependencies are shared by both Understand and IDEA.

Example (1): …\hadoop\fs\**TestHDFSFileContextMainOperations**.java →
…\hadoop\hdfs**\DistributedFileSystem.**java

```
import org.apache.hadoop.hdfs.DistributedFileSystem;
            …
DistributedFileSystem fs = cluster.getFileSystem();
```

Example (2):
…\hadoop\mapreduce\counters\**CounterGroupBase**.java→
…\hadoop\mapreduce**\Counter.**java

```
import org.apache.hadoop.mapreduce.Counter;
            …
Public interface CounterGroupBase<T extends Counter>
```

Both tools successfully demonstrate that they detect dependencies via import statement, attribute declaration, object initialization, method call or passed as parameter. These are very typical dependency relationships.

Understand Only

Here are the most significant examples of dependencies existing only in Understand.

Example (1): …\src\main\java\org\apache\hadoop\crypto\random\**OsSecureRandom**.java→
…\src\test\java\org\apache\hadoop\crypto\**TestCryptoCodec**.java

The depended class is TestCryptoCodec under the test folder. Surprisingly, we found 1220 out of 11375 Understand only dependencies whose depended class is TestCryptoCodec. However, when we investigate the source code, we could not find the dependency relation. In addition, it also violates our intuition that class under \main folder should not call a class under \test folder. We tend to recognize it as dependency error.

Example (2): …\hadoop\yarn\server\resourcemanager\**ClientRMService**.java →
…\hadoop\yarn\api\protocolrecords\**CancelDelegationTokenResponse**.java

```
import org.apache.hadoop.yarn.api.protocolrecords.CancelDelegationTokenResponse;
            …
@Override

  public CancelDelegationTokenResponse cancelDelegationToken(
```

If a method is annotated as @Override, all the classes invoked (dependency relation) inside the method will not be detected by IDEA. However, they will be detected by Understand. This is a weakness of IDEA.

IDEA only

Here is an example of a dependency that exists only in IDEA (i.e. unique to IDEA).

Example (1): …\hadoop\security\token\delegation\ZKDelegationTokenSecretManager.java→ …\hadoop\io\Writable.java

Delegation key = new DelegationKey();

Public class DelegationKey implements Writable legationg

IDEA includes parent class dependency, while Understand does not.

Limitation

There is limit to the dependency extraction and analysis process employed. For one, sampling is chosen to make the process scalable. However, there is a possibility that a critical point is missed since not all classes are examined.

Alternative Extraction Tool: srcML

We initially considered extracting dependencies using srcML. It is a toolkit that can construct XML-representation of a given source code. Through its query command 'xpath', the user can choose the type of dependency to be extracted. We planned to extract dependencies by querying the class attribute/variable declaration statement. However, it is rather challenging as we will need to extract the classes and its associated filepath, and compile the dependency based on the import statement. Since the import statement is required, the result of dependency extracted in this way will be like that of Include. Hence, we decided not to proceed with this plan.

Lessons Learned

Through the creation of this report, we've learned some key aspects about extracting dependencies in Hadoop:

- Powerful extraction tools like Understand can make errors, as seen in the qualitative analysis
- Why some files depend on another can only be seen after thorough digging through classes and methods.
- Studying dependency reveals a substantial part of the design of the whole system
- There are many approaches to extracting dependencies (import, method calls and object initializations, java annotations, inheritance and implementations, etc

Conclusion

Extracting Hadoop's dependencies allowed us to analyze and understand how each extraction tool works. We closely examined Understand by SciTools, Intellij IDEA, and a custom program made by Reverse 'em All called Include that only detects import statements. By comparing Understand to both IDEA and Include, we could explain why some dependencies were exclusive only to one tool, and while some dependencies were detected by both tools. We found that a consumer product like Understand could make mistakes.

As to which tool we would choose if we had to choose one, we would choose the alternative tool srcML. SrcML has the versatility of extracting a more exact type of dependency user would like to examine, and hence more helpful in learning the structure of a software system.

Data Dictionary

Include: a java program to output the list of filepath of java classes in a given root directory, with the filepath of the classes it declared in its import statements.

Comparison.java: a java program that take two dependency files and output a text file containing which dependencies occurred in both files, occurring only in one of the files, and dependencies occurring only in the other file.

Understand: a static analysis tool focused on source code comprehension, metrics, and standards testing

calls: method calling / invoking another method to execute its functionality

Inheritance: the system where a class inherit another class, thereby inherited the properties and methods as well

Object initializations: the statement whereby a declared variable is initialized to a reference or primitive value

Implementations: the body of a class or a method of a class

Overrides: the act of overriding a parent's method by the inherited class

Throws: the act of throwing a throwable object in cases of exception

References:

[1]*Understand 4.0 User Guide and Reference Manual*, 1st ed. St. George: Scientific Toolworks, 2015, p. 13.