Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

THOMAS WOLTER
BACHELOR THESIS

# A COMPARISON STUDY OF OPEN SOURCE LICENSE CRAWLER

Submitted on 13 May 2019

Supervisors:
Prof. Dr. Dirk Riehle, M.B.A., Michael Dorner, M. Sc
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander-Universität Erlangen-Nürnberg

# Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

---

Erlangen, 13 May 2019

# License



---

Erlangen, 13 May 2019

# Abstract

In order to include open source software in a project, a software developer must abide to the license the software is published under. However, there are no clear guidelines for license placement in open source projects. As a result, the location of the relevant licensing text can vary for each project, making the license identification process a difficult task. A proposed solution for this issue are license crawlers designed to search project directories for the licensing information. This thesis aims to evaluate the existing license crawlers for their functionality and performance, in order to find out if a sufficient solution to the problem exists.

To do so, we performed a two phased benchmark with 6 license crawlers and 75 open source projects. Firstly, we determined which of the software tools found the most licenses in a direct competition. Secondly, we evaluated conflict situations in the output of the best performing license crawlers.

Our results show, that FOSSology and Scancode performed the most reliably. Looking at the conflict situations, we also determined that FOSSology made fewer errors in its evaluation. However, we also found that there are four error categories the crawlers are especially susceptible to.

# Contents

# 1 Introduction

## 1.1 Original Thesis Goals

The goal of this thesis was to survey existing license crawlers on their workflow and their effectiveness at identifying licenses in open source projects. To do so, a benchmark with the top 1000 most starred projects on GitHub was planned. By comparing the results of each crawler we wanted to find out which license crawler performs best.

## 1.2 Changes to Thesis Goals

The amount of projects for the benchmark was changed from the top 1000 to a sample of 75. The reason for this was the project quality among the top rated projects. A detailed explanation can be found in chapter 2.4.2.

# 2 Research

## 2.1 Introduction

When being faced with a challenge, today's software developer can oftentimes fall back on a variety of open source projects. These projects can offer preexisting solutions to the task at hand and a chance to access the knowledge and work of one's peers. However, while doing so, it is of importance to consider the license a project is published under. The license determines what restrictions are placed on the usage of a specific software (Lerner & Tirole, 2005). Therefore, a software developer should always determine the license of an open source project and only use it if he is willing to abide to the imposed limitations. However, the process of finding the appropriate licensing information often poses a problem.

There are no clear guidelines on where exactly the licensing text should be placed. As a result, the location can vary depending on the project. GitHub, a popular software development platform, for example suggests that a dedicated file should be included in the root directory. These files are often named 'LICENSE' or 'COPYING' and contain only license relevant data. Another suggestion is including the license text in the 'README' file. However, these two suggestions are only described as best practices and far from being the only locations used in open source projects.[1] Thus, in the worst case, a project can use completely unknown means to indicate the license. Furthermore, projects often already use the work of others and therefore contain several licenses in different locations of the directory tree, potentially resulting in a conflict situation between licenses (Rosen, 2005). As a result, identifying the licenses of larger projects can become a time intensive and error-prone task.

However, there are already a variety of proposed solutions to this problem. Several license crawler aim to automatically scan project directories for possible licensing texts. This process is supposed to be significantly faster than manually inspecting every subdirectory by hand. Upon completion, the user is then

[1] https://help.github.com/en/articles/licensing-a-repository

presented with an overview of the results. However, the crawlers have not been thoroughly tested yet. Thus, this thesis aims to evaluate the existing crawlers.

## 2.2 Related Work

The necessity of identifying an open source projects license is well documented. If a developer has the intention to open source his work, a license must be chosen. Generally speaking, the project owner must determine how restrictive the project is going to be. On the one hand, there are strongly restrictive licenses. With these licenses, all changes must always be published under the same license as the original. On the other hand, there are more permissive licenses . These licenses give potential users more freedom of use (Lerner & Tirole, 2005). Overall, the choice of restrictiveness can influence aspects of the development process in meaningful ways. Stewart, Ammeter and Maruping (2006) for example suggest, that contributors tend to contribute more, the less restrictive the licensing is. Furthermore, it is important to consider that combining licenses of different types may lead to a conflict situation (Rosen, 2005). Thus, identifying the license of an open source project is of great importance. However, the process of actually identifying where the licensing information can be found, is not covered in much detail.

Vendome et al. (2017) suggested that there are two commonly used ways to declare licenses in open source projects. Firstly, source code file often contain a license declaring comment at the top of the file. Secondly, developers also add dedicated files containing the licensing text to the projects directories.

German, Manabe and Inoue (2010) investigated the difficulties of identifying open source licenses in source code. They also concluded that the needed data is most commonly found in a comment at the top of code files, but a variety of challenges make an algorithmic solution to this problem a difficult task. They split these challenges are in 3 categories:

- ' Finding the license statement':

  Covers the problems that arise because no standard practices about license placement exist. This includes the necessary text being mixed with unrelated text and files having multiple licenses.

- ' Language related':

  Covers the problems that arise because of human errors in writing/copying the license text. This includes spelling errors and grammar changes.

- ' License customization':

This covers the problems that arise when licenses are customized by the project owner.

Based on their research, they suggested that license verification tools should be created for the development process. By doing so, a more standardized approach at license inclusion could be achieved.

## 2.3 Research Question

Overall, the research goal was to find out, if a reliable license crawler already exists. The desired tool should be able to scan an open source project and find all licenses contained within. However, properly validating whether a crawler located every possible license is a difficult task. Therefore, the two research questions we aim to answer compare the existing crawlers among each other.

Firstly, RQ1 surveys all the existing crawlers for their output and general correctness. We wanted to investigate which crawler exceeds at finding licenses in a project and which of the software tools underperforms in a direct comparison.

**RQ1:** ***How do the existing license crawlers compare at finding licenses in open source projects?***

Secondly, RQ2 aims to follow up to this question. We wanted to find out what differences there are between the best performing crawlers on a file-by-file basis.

**RQ2:** ***Is there a significant difference in output between the top license crawlers?***

## 2.4 Research Approach

### 2.4.1 Overview

In order to answer our research questions, we oriented ourselves towards Stol and Fitzgeral (2018) and their ABC framework. The research approach was supposed to maximize the potential for generalizability of our findings on license crawlers. Thus, we chose to conduct a sample study. To do so, we first sampled existing license crawlers and open source projects. This selection then served for a 2 phased benchmark of the software tools against each other.

### 2.4.2 Sampling

#### Sampling of crawlers

Overall, there is already a good amount of crawlers that attempt to solve the problem of license identification. The goal of this thesis was to consider as many candidates as possible. Thus, an internet search was conducted. Searching for keywords like 'license crawler', 'license identifier' and 'license detector' on popular online platforms such as Google and GitHub yielded a variety of results. Additionally, some of the identified crawler's descriptions gave recommendations to other, similar projects. These software tools were then also taken under consideration.

Once a potential crawler was found, it was checked for the following criteria:

1. A function to scan a given project for licensing information. This can be limited to the root directory or extend to the entire directory tree. Only looking at a single file however was considered to be insufficient.
2. The scanning process is mostly automated and does not require much input beyond an initial directory name or input file.
3. The output presents the results in a comprehensive manner. In order to make more in-depth comparisons in phase 2 of our benchmark we needed the crawlers to give details about their finds.
4. The project is open sourced.

If all criteria were fulfilled, the crawler was considered for further evaluation in phase 1 of our benchmark. The reasoning for setting the requirements so broad, is that the field of research is still relatively new. While a number of crawlers already exist, little has been done to document which works best. As a result, we wanted to make sure any valid license crawler was taken under consideration.

#### Sampling of projects

The sampling of projects was one of the aspects of this thesis that was changed during the development phase. Originally, we intended to benchmark the selected crawlers by using the 1000 most starred projects on GitHub. The platform was chosen, as it hosts a large number of projects and offers a curation aspect in the form of 'starring' a project. By using the starring feature, users can bookmark a project. The GitHub documentation states that the overall number of stars indicates the level of interest in the project.[2] The problem we faced was, that

[2]https://developer.github.com/v3/activity/starring/

the interest level in a project did not necessarily mirror the factors we desired in a project. We were looking for projects that software developers might make direct use of while working on software. However, the most starred section of GitHub does containing a variety of projects that do not fit this pattern. The section includes many projects that are not supposed to be integrated in other projects. Overall, we wanted to avoid using projects that cover:

- Project lists that accumulate links to different programming resources. An example for this is the most starred repository on the platform with about 300.00 stars. 'freeCodeCamp' is an open source codebase that is aimed at teaching beginners to code.
- Book collections related to specific topics. There are for example collections of freely available programming books among the most starred projects.
- Joke projects that have no real life application.
- Foreign language projects. As we wanted to investigate the projects on a directory tree level, we needed to be able to understand the files contained in a project.

To counteract this problem, we designed a method to filter out the undesired projects. In order to be added to the final test set, a software project had to fulfill the following criteria:

- The project is actively working on developing code.
- The project is providing more than just links to other resources.
- It must be feasible that the project is implemented in other projects.

### 2.4.3 Study Design

#### Phase 1

Firstly, to find out if any of the license crawler significantly underperformed, we chose to benchmark all sampled software tools with the success criteria 'total licenses found' and 'unique licenses found'. As little research has been done in this field of research, no gold standard for comparison exists. Thus, we were unable to compare the output against a perfect outcome. Instead we chose to benchmark the crawlers against each other.

To extract as much data as possible, we tested each crawler with each project in our test set. This entire process was almost entirely automated. All license crawler besides 'FOSSology' were run as a command line application. This allowed us to start a simple shell loop, that started each crawler with our entire test

set. Additionally, we used the 'time'-command available in shell to record the scanning duration of each crawler. The output of 'go-license-detector', 'askalono' and 'licensee' were then saved by writing the command line output to a text file. For 'Scancode' and 'licensechecker' we simply saved the corresponding output files for each project. In order to use FOSSology the respective client was used and each project of our test set was uploaded manually. We then started the 'nomos' scanner. After the completion of the scans, the 'DEP5'-file and the 'SPDX tag:value'-file were download to extract the necessary information for both phases.

To find the total amount of licenses, each output was crawled with a python script. The total number of licenses each crawler found then served as a base for our comparison.

To best display the unique licenses found per project we chose do create a simple table view. This enabled us to make an easy side-by-side comparison. The leftmost column of the table is used to indicate the project. The topmost row of the table shows which license crawler the result belong to. This helped us easily recognize which license crawler performed worse than the others in a side-by-side comparison. To extract the necessary information from the output, we wrote a python script that crawls the different output files we created while benchmarking. The script scrapped the data and then automatically place the data in a CSV file to create the table view. Figure 2.1 gives an exemplary overview of the table with 2 projects and 2 crawlers. The actual output of the script contained all projects and all crawlers, but is not shown in this thesis due to the size.

| | **Crawler 1** | **Crawler 2** |
|---|---|---|
| **Project 1** | License 1<br>License 2 | License 1<br>- |
| **Project 2** | License 1<br>License 2<br>- | License 1<br>License 2<br>License 3 |

**Figure 2.1:** Example of the table view with 2 projects and 2 crawlers

After the creation of the table, we compared the results with each other in order to eliminate the weakest performing crawlers. In the end, a list of the best performing crawlers was passed to phase 2 of our benchmark.

### Phase 2

Upon completion of phase 1, we wanted to compare the best crawlers in a more in-depth manner. To do so, the outputs from the initial benchmark were taken

under consideration again. This time we looked at the exact locations of license hits in the directory tree. By comparing the results of each crawler on this level of depth, we were able to find conflicts between the remaining tools and thus draw conclusions. As there are a total of 292573 files in our test set, it was not possible to check every conflict that occurred by hand. Instead, we looked at 25 random conflict situations.

To display the results in a comprehensive manner, we designed a python script to create a directory tree view for every project in our test set. In these files we displayed the path to every component of the project. Each of these paths was then marked with the corresponding result of the remaining license crawlers. Figure 2.2 shows an exemplary output.

| Root directory | Subdirectory 1 | Subdirectory 2 | Crawler 1 | Crawler 2 |
|---|---|---|---|---|
| File 1 | | | License A | License A |
| File 2 | | | - | - |
| Directory 1 | File 3 | | - | - |
| Directory 1 | File 4 | | - | - |
| Directory 2 | File 5 | | - | - |
| Directory 2 | Directory 3 | File 6 | License B | License C |

**Figure 2.2:** Example of the directory tree view with one license conflict

Afterwards we searched each of the files for conflicting evaluations of the license crawlers. If two crawlers came to a different conclusion, we added the path and the results to a master file containing every license conflict in the entirety of our test set. Upon being added to the master file, we also assigned each path a number for identification purposes. In order to select files at random, we generated 25 random numbers. These numbers corresponded to the ones assigned in the master file. The selected files were then investigated by hand.

While looking at specific files, we first located the relevant licensing information. Afterwards, we compared our result to the evaluation of the license crawlers. By doing so, we were able to find out which crawler was in the wrong. This allowed us to draw conclusion on the overall correctness of each crawler. Furthermore, the cases in which both crawlers failed gave us information on potential error sources.

## 2.5 Research Results

### 2.5.1 Sampled crawlers

The following gives a short overview of the workflow of the sampled crawlers. Additionally, a sample output is demonstrated for each crawler by running the software tool on a small demo project we created.

#### askalono

Askalono is a license crawler written in the Rust programming language. To identify possible licenses, two commands are available. On the one hand, the user can input the 'id'-command and a file path. The crawler then scans only the given file for licensing information. On the other hand, inputting the 'crawl'-command and a directory path, scans the entire directory tree. The workflow of the id-command is as follows:

1. Normalize the file

   Elements such as whitespaces are not necessarily important for the comparison process. Thus, redundant aspects of the input file are removed. The end result of this step is a normalized text version of the input file.

2. Apply the algorithm

   This normalized output is split into bigrams. The resulting set of word pairs is then checked with sets of bigrams created from actual licenses. After all comparisons are concluded, askalono outputs the top result.

If the crawl command is used, this workflow is repeated for all possible files.

The output lists the path to every analyzed file and the crawler's end result. In addition, each file receives a confidence value, describing the similarity of the bigram sets with a Sørensen-Dice coefficient. This coefficient describes the overall similarity of the sets.[3] In Listing 2.1, an exemplary output of the crawl function is displayed.

```
$ askalono/askalono.linux crawl demo-master/
demo-master/LICENSE
License: MIT (original text)
Score: 0.994
demo-master/subdirectory/LICENSE
License: MIT (original text)
```

[3] https://github.com/amzn/askalono

```
Score: 0.994
```

**Listing 2.1:** Exemplary output: askalono

### FOSSology

FOSSology is a toolkit, that provides the user with license and copyright scanners, as well as further tools enabling license compliance. However, for this thesis only the license scanner aspect will be considered. It is available as a command line application and a client version. The client provides reports in more detail and was therefore chosen for our benchmark. The workflow of the 'nomos' scanner is as follows:

1. Search for keywords

   The crawler attempts to search the projects files for specific keywords that often indicate license texts. Furthermore, in FOSSology there is a measure to avoid false positives. In addition, the tool considers heuristics during this process. For example, certain sentences should appear either close to each other or not together at all. The found files are then passed to the actual scan.

2. Apply algorithm

   'Nomos' uses a regular pattern algorithm for it's identification process. Upon completion the results are made available to the user in the form of short overviews or lengthy reports.

The more in-depth reports give the exact location of each license identified, while the short overview simply gives a list ordered by number of occurrences.[4] Figure 2.3 gives a look at the overview.

| Scanner Count | Concluded License Count | License Name |
|---|---|---|
| 2 | 0 | MIT |

**Figure 2.3:** Example of the output in the FOSSology client

[4]https://www.fossology.org/features/

### go-license-detector

As the name suggests, the 'go-license-detector' is written in the Go programming language. The tool is offered both as a library and as a command line application. To determine what licenses a project may hold, the user gives the crawler a project directory. The crawler then splits its workflow in several steps:

1. Sample out files with high potential

   The crawler scans the given directory for the files containing the licensing information. It mostly looks for LICENSE files. The results passed to the next step are the identified files.

2. Normalize the found files

   To ease the identification process, the file's have several parts of their content stripped away. Firstly, the original file's format is converted to plain text, and the unnecessary content, such as HTML tags, is removed. Secondly, the crawler strips away information, which does not impact the end result. Lastly, the crawler erases punctuation from the file. By removing these aspects, as little necessary information as possible is lost. The result passed to the next step are the files in their normalized form.

3. Apply algorithm

   The go-license-detector creates unigrams of the normalized text. These are then used to count the occurrences of each word. By using 'Weighted MinHash' and 'Locality Sensitive Hashing' the crawler picks similar licenses. Then a value is set for the similarity.

A possible exception to this pattern, is the absence of 'LICENSE'-files. If no such files are present, the search is expanded upon. The crawler starts looking for possible alternatives, such as README files. This is a rather difficult task, as there are countless possible naming conventions for these files. To solve this problem, the development team has designed a collection of common expressions aimed to identify them as best as possible. It important to consider that a README file often contains more than just licensing information, so the file is searched for the passages that contain possible license names.[5]

Upon completion, the crawler will output a summary of the licenses found. This includes a percentage, describing the similarity to the identified license as well as the actual name of the license. Listing 2.2 provides an exemplary output of command line execution:

```
$ license-detector demo-master/
demo-master/
```

[5]https://github.com/src-d/go-license-detector

```
99% MIT
93% JSON
85% MIT-feh
82% Xnet
```

**Listing 2.2:** Exemplary output: go-license-detector

It is important to note, that the go-license-detector focuses on the directory it was given by the user. It does not search the sub directories for potential information. To be able to compare it with other crawlers that are able to do so, every directory of every selected project was tested for the later part of this thesis.

#### licensechecker

Licensechecker is written in the Go programming language It is available as a command line application. To start the crawler, it requires a directory path. The resulting workflow can be split in 2 steps:

1. Scan the parent directory for license files

   The crawler attempts to find a license file in the input directory by looking for specific keywords in filenames.

2. Apply algorithm on results

   To identify which license the file contains, licenschecker compares ngrams in the range 2 to 8 of the actual license texts and the found files. If a license was found, the result is then checked again with the 'Vector Space Modell'. Furthermore, a confidence value describing the similarity is assigned to each hit. This license will then be used for every file in the subdirectories of the project.

3. Remember the results and proced to the subdirectories

   The result is then memorized and passed on to the subdirectories. The crawler also scans each subdirectory for further licenses. The same algorithm as step 2 is used to identify potential hits. If a new license is found, it is also pased on to the following subdirectories. This is repeated until a leaf in the directory tree is found. In the end, each file is marked with every license found on the path.

The output file shows the entire directory tree and marks every file with a coresponding license and a confidence value.[6] Figure 2.4 gives a shortend exemplary output in the CSV format.

[6] https://boyter.org/2017/05/identify-software-licenses-python-vector-space-search-ngram-keywords/

| filename | directory | license | confidence |
|---|---|---|---|
| LICENSE | demo-master | MIT | 1.00% |
| file1 | demo-master | MIT | 1.00% |
| file2 | demo-master | MIT | 1.00% |
| LICENSE | demo-master/subdirectory | MIT | 1.00% |
| file3 | demo-master/subdirectory | MIT | 1.00% |

**Figure 2.4:** Exemplary output: licensechecker

### licensee

Licensee is written in the Ruby programming language. The tool is available both as a command line application and a library. License identification is possible with a 'detect'-command, that takes directories, files or a GitHub repository as input. The resulting workflow looks as follows:

1. Sample out files with high potential

   The crawler attempts to estimate the most likely files containing the licensing text. To do so, a number of regular expressions are saved and used to compare to each file name.

2. Normalize the found files

   The software tool removes whitespaces and copyright notices from the file to make the comparison step easier.

3. Apply Algorithm

   Firstly, an attempt at an exact match is made. The developers state that comparisons are easy for Ruby to handle. Secondly, if there was no match, the tool will attempt to find out if a file is at least similar to an existing one. The software employes a Sørensen-Dice coefficient to describe the overlap between the normalized text and a license text.

The output then list every file found and the result, including the license found and the confidence.[7]

```
$ licensee detect demo-master/
License: MIT
Matched files: LICENSE
LICENSE:
  Content hash: 46cdc03462b9af57968df67b450cc4372ac41f53
  Confidence: 100.00%
  Matcher: Licensee::Matchers::Exact
  License: MIT
```

[7] https://github.com/licensee/licensee/blob/master/docs/what-we-look-at.md

**Listing 2.3:** Exemplary output: licensee

### scancode

Scancode is written in the python programming language. It can be used as a command line application or as a library, in order to 'discover and inventory open source and third-party packages'.The crawler takes a directory path and an output file as input for the following workflow:

1. Gather every file in the directory and subdirectories. Afterwards, the files are grouped in different categories depending on their type.
2. Start the scan on the collected files, by comparing the texts to a search index. This includes a collection of possible references to licenses in the form of 'thousands of license texts, notices and examples'. Hits are then written to the output file.[8]

The software can output the result in JSON, HTML, CSV and SPDX. The output lists every file and all identified licenses as shown in figure 2.5.

Copyrights and Licenses Information

| path | start | end | what | value |
|---|---|---|---|---|
| demo-master/LICENSE | 3 | 7 | license | mit |
| demo-master/subdirectory/LICENSE | 1 | 7 | license | mit |

**Figure 2.5:** Example of the output of scancode

### Others

Besides the previously mentioned tools, 2 more were discovered. However, they did not meet the criteria we imposed for our sampling:

- 'licenseclassifier': Can only be used to scan single files.
- 'LiD': We were not able to get the tool to work on our system.

[8]https://github.com/nexB/scancode-toolkit/wiki

### 2.5.2 Sampled projects

We gathered a total of 75 projects fitting the criteria we imposed during our planning. This collection was then used as a test set for the benchmarking. A complete list can be found in appendix A.

### 2.5.3 Phase 1

#### Total licenses found

Figure 2.6 represents a first overview of our benchmarking. It shows the total amount of license hits by each of the crawlers excluding licensechecker. Licensechecker had to be excluded, because of it's output. The crawler marks the entire directory if a license was found and gives no information on which file contains the licensing text. Thus, we can not determine how many hits took place in total. Overall, Scancode performed the best with a total of 82105 hits and an average of about 1094 licenses per project. FOSSology came in second with 67779 hits and about 903 average hits. Go-license-detector, askalono and licensee are far behind those with 1953, 1144 and 131 total licenses found.

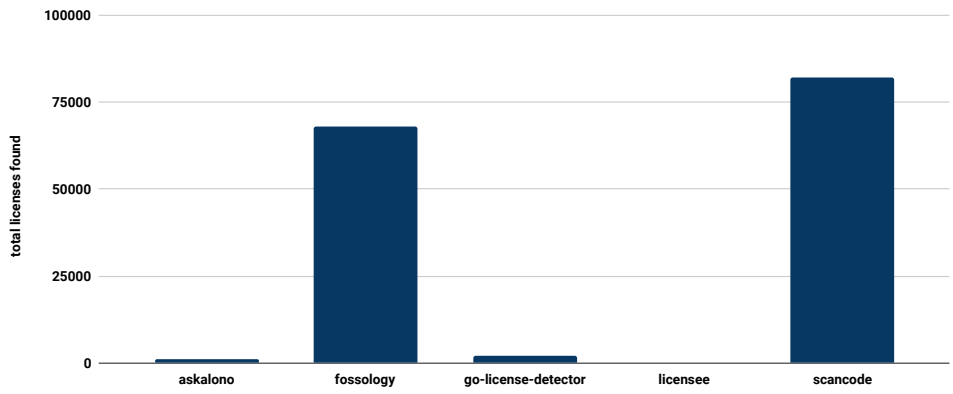


**Figure 2.6:** Total licenses found

#### Unique licenses found

Furthermore, we considered the amount of unique licenses found in each project. Like with our previous success criteria, Scancode found the most unique licenses. A total of 725 unique hits were registered while scanning the 75 projects. FOSSology came in second, finding 667 licenses. go-license-detector, licensechecker

and askalono found 361, 238 and 136 licenses respectively. Lastly, licensee found only 58 unique licenses.

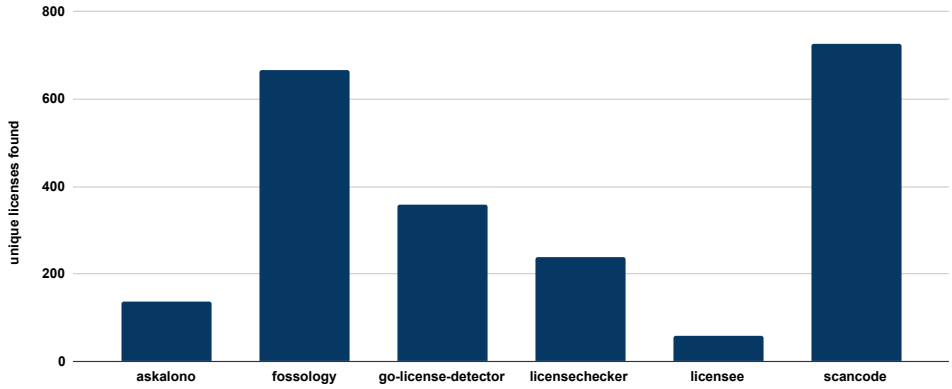


**Figure 2.7:** Unique licenses found

### Time spent scanning

Furthermore, we also noted the total time spent scanning our test set for all license crawlers. In this regard, go-license-detector took the longest. However, it is important to mention that it is not designed to scan the entire directory tree. As a result the process is not optimized. The best performing license crawlers FOSSology and Scancode were matched pretty even. The former scanned for 11367.739 seconds and the later scanned for 13370.398 seconds.

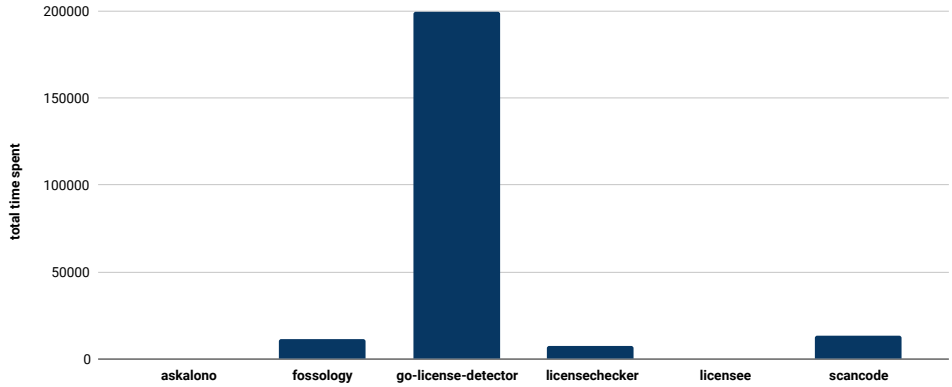


**Figure 2.8:** Total time spent scanning

Because of the large differences in output and efficiency we discovered during our benchmark, we chose to focus on Scancode and FOSSology going into phase

2. Thus, askalono, licensee, licensechecker and go-license-detector were removed from our list of crawlers.

### 2.5.4 Phase 2

A total of 149884 total license hits occurred while scanning with Scancode and FOSSology. Overall, the two license crawler agreed on their result in 124756 of those cases. This amount to a total of 83.24% of all license hits. However, there were a total of 12564 conflict situation. In order to analyze these conflict situations, we picked 25 random cases to cover in more detail. Table 2.1-2.3 show the results of Scancode and FOSSology for these cases, as well as the actual result we found by checking the files ourselves. Furthermore, we marked the result based on the quality. A green marking means the evaluation was correct, a yellow marking means the result is a corner case and the red marking means an error occurred.

| Conflict | Fossology result / Evaluation | Scancode result / Evaluation | Licenses found | License relevant text / Evaluation |
|---|---|---|---|---|
| 1 | LicenseRef-UnclassifiedLicense<br>Only a reference was marked. | MIT<br>The result is based on insufficient information. | MIT License | "Use of this source code is governed by an MIT-style license that can be found in the LICENSE file at https://angular.io/license"<br>⇒ The statement is not definitive. A crawler would have to check the provided link (None of the crawler do). |
| 2 | LicenseRef-UnclassifiedLicense<br>Only a reference was marked. | MIT<br>The result is based on insufficient information. | MIT License | "Use of this source code is governed by an MIT-style license that can be found in the LICENSE file at https://angular.io/license"<br>⇒ The statement is not definitive. A crawler would have to check the provided link (None of the crawler do). |
| 3 | MIT<br>Correct evaluation | BSL-1.0 , MIT<br>False flag | MIT License | "Distributed under the MIT software license, see the accompanying file COPYING or http://www.opensource.org/licenses/mit-license.php."<br>⇒ All files in COPYING point to MIT / Original statement sufficient |
| 4 | MIT<br>Correct evaluation | BSL-1.0 , MIT<br>False flag | MIT License | "Distributed under the MIT software license, see the accompanying file COPYING or http://www.opensource.org/licenses/mit-license.php."<br>⇒ All files in COPYING point to MIT / Original statement sufficient |
| 5 | MIT , GPL-2.0 , LicenseRef-Dual-license<br>Correct evaluation | MIT , GPL-2.0<br>No mention of dual licensing | Dual licensed under the MIT or GPL-2.0 licenses. | "Dual licensed under the MIT or GPL Version 2 licenses. http://jquery.org/license" |
| 6 | MIT , LicenseRef-CC-BY-SA<br>False flag | MIT , CC-BY-SA-2.5<br>False flag | MIT License | Full license text: MIT License : "Dokumentasi dan logo grafis MDN "ABOUT_TEXT_MDN_DOCS" berlisensi Creative Commons Attribution, <a href='{MDN_DOCS_LICENSE}'>CC-BY-SA 2.5 Unported</a>."<br>⇒ Not actual licensing information |
| 7 | LicenseRef-MIT-CMU-style<br>Modified license | HPND<br>Modified license | Modified License | Slightly modified license text |
| 8 | LicenseRef-Python<br>Correct evaluation | GPL-2.0-plus<br>False flag | Python Software Foundation License | "This module is free software, and you may redistribute it and/or modify it under the same terms as Python itself, so long as this copyright message and disclaimer are retained in their original form." |
| 9 | Apache-2.0<br>Correct evaluation | Apache-2.0 , BSD-3-Clause<br>False flag | Apache-2.0 | "Copyright 2010-2018 JetBrains s.r.o. Use of this source code is governed by the Apache 2.0 license that can be found in the license/LICENSE.txt file."<br>"Copyright 2000-2018 JetBrains s.r.o. Use of this source code is governed by the Apache 2.0 license that can be found in the license/LICENSE.txt file." |
| 10 | LicenseRef-BSD<br>Only a reference was marked. | BSD-3-Clause<br>The result is based on insufficient information. | BSD-3-Clause | "Copyright (c) 2011, the Dart project authors. Please see the AUTHORS file for details. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file." |

**Table 2.1:** Conflict situations 1-10

| Conflict | Fossology result / Evaluation | Scancode result / Evaluation | Licenses found | License relevant text / Evaluation |
|---|---|---|---|---|
| 11 | CC-BY-SA-4.0<br>Correct evaluation | CC-BY-4.0<br>Incorrect version | CC-BY-SA-4.0 | This Meteor Code of Conduct is licensed under the [Creative Commons Attribution-ShareAlike 4.0 International](https://creativecommons.org/licenses/by-sa/4.0/) license. This Code was last updated on August 28, 2017. |
| 12 | LicenseRef-BSD<br>Only a reference was marked. | BSD-3-Clause<br>The result is based on insufficient information. | BSD-3-Clause | "Use of this source code is governed by a BSD-style license that can be found in the LICENSE file."<br>⇒ The statement is not definitive. A crawler would have to check the file. |
| 13 | BSD-3-Clause<br>Correct evaluation | None<br>Miss | BSD-3-Clause | Full license text (no header): BSD-3-Clause |
| 14 | LicenseRef-BSD<br>Only a reference was marked. | BSD-3-Clause<br>The result is based on insufficient information. | BSD-3-Clause | Use of this source code is governed by a BSD-style license that can be found in the LICENSE file.<br>⇒ The statement is not definitive. A crawler would have to check the file. |
| 15 | LicenseRef-BSD<br>Only a reference was marked. | BSD-3-Clause<br>The result is based on insufficient information. | BSD-3-Clause | Use of this source code is governed by a BSD-style license that can be found in the LICENSE file.<br>⇒ The statement is not definitive. A crawler would have to check the file. |
| 16 | LicenseRef-BSD<br>Only a reference was marked. | BSD-3-Clause<br>The result is based on insufficient information. | BSD-3-Clause | Use of this source code is governed by a BSD-style license that can be found in the LICENSE file.<br>⇒ The statement is not definitive. A crawler would have to check the file. |
| 17 | LicenseRef-Unicode-TOU<br>Correct evaluation | Unicode Inc License Agreement<br>Correct evaluation | Unicode® Copyright and Terms of Use | License & terms of use: http://www.unicode.org/copyright.html |
| 18 | LicenseRef-Unicode-TOU<br>Correct evaluation | Unicode Inc License Agreement<br>Correct evaluation | Unicode® Copyright and Terms of Use | License & terms of use: http://www.unicode.org/copyright.html |
| 19 | LicenseRef-GPL<br>Only a reference was marked. | GPL-2.0+, GPL-1.0+<br>False flag, Miss | Zlib | 42. The match.asm code in contrib is under the GNU General Public License. Since it's part of zlib, doesn't that mean that all of zlib falls under the GNU GPL?<br>=> Actual license found in different file. Linked in the FAQ. Almost impossible to find. |
| 20 | GPL-2.0+ , LGPL-2.1+ , MPL-1.1 , LicenseRef-Dual-license<br>Dual-license not mentioned in | GPL-2.0+, LGPL-2.1+, MPL-1.1<br>Correct evaluation | GPL-2.0+, LGPL-2.1+, MPL-1.1 | Version: MPL 1.1/GPL 2.0/LGPL 2.1 |

**Table 2.2:** Conflict situations 11-20

| Conflict | Fossology result / Evaluation | Scancode result / Evaluation | Licenses found | License relevant text / Evaluation |
|---|---|---|---|---|
| 21 | LicenseRef-UnclassifiedLicense<br>False flag | None<br>Correct evaluation | None | "Names should be added to this file only after verifying that the individual or the individual's organization has agreed to the appropriate Contributor License Agreement, found here:" |
| 22 | LicenseRef-MIT-BSD<br>Only a reference was marked. | MIT , BSD-3-Clause<br>The result is based on insufficient information. | Unclear license statement | Modernizr 2.8.3 (Custom Build) \| MIT & BSD |
| 23 | LicenseRef-BSD-possibility<br>Incorrect version | BSD-3-Clause<br>Correct evaluation | BSD-3-Clause | License: BSD 3 clause |
| 24 | Apache-2.0<br>Incorrect version | Apache-2.0 , Public Domain<br>Incorrect version, False flag | Apache License v2.0 with Runtime Library Exception | Apache License v2.0 with Runtime Library Exception<br>"This file is based on the reference C implementation, which was released to public domain by:" |
| 25 | LicenseRef-Apache-possibility<br>Incorrect version | Apache-2.0<br>Correct evaluation | Apache-2.0 | licenses(["notice"])  # Apache 2.0 |

**Table 2.3:** Conflict situations 21-25

Evaluating each of the results on the correctness compared to our conclusion, we created the following tables. We ignored cases that had insufficient licensing information for FOSSology, because the crawler only marked these with a Reference-tag. The tag is supposed to prompt the user to act and clarifies that the crawler only found a small hint at a license. As a result, it is difficult to judge whether the judgement was correct or not. Scancode had many false positives, marking license hits that were not actually present or had insufficient data. Overall, the positive prediction value was 37.5%. FOSSology on the other hand stayed neutral in a lot of cases. Thus, it's positive prediction value was much higher with 66.66%. Both crawlers only had few complete misses.

| | **license present** | **license not present** |
|---|---|---|
| **license identified** | 12 | 20 |
| **license missed** | 6 | 1 |

**Table 2.4:** Confusion matrix: Scancode

| | **license present** | **license not present** |
|---|---|---|
| **license identified** | 14 | 7 |
| **license missed** | 4 | 0 |

**Table 2.5:** Confussion matrix: FOSSology

## 2.6 Results Discussion

The results of phase 1 of our benchmark showed that go-license-detector, licensechecker, askalono and licensee underperformed by a significant margin. Overall, the four crawler's combined total hits only amounted to 16.76% of all hits. Looking at the workflow of each crawler and the results of phase 2, we can see that the crawlers do not look at all files. They make preselections of files they deem likely to have relevant data. FOSSology and Scancode check every single file available and ultimately provide a better result. Furthermore, every conflict situation we investigated dealt with non license dedicated files. This finding also matches statements of German et al. (2010) about license text often being found at the top of coding files, as the four crawler do not look at this aspect.

The results of phase 2 helped us identify a variety of flaws in both remaining crawlers. In most of the cases we looked at, a conflict between the two crawlers actually meant the result of both was either erroneous or insufficient. Overall,

by analyzing the 25 conflicts, we were able to determine 4 different categories of flaws. The following will give a short explanation of the error and the handling of the problems by Scancode and FOSSology.

**1. License references**

One of the major issues was the handling of short references in files. Oftentimes the header containing the licensing information did not contain the actual relevant data, but pointers to the actual location. Case 12 for example states that the file is licensed under a 'bsd-style'-license. To find the actual license one has to look at the LICENSE file. However, this file can only be found several directories above the original file. By checking manually we found out that the project is licensed under the BSD-3.0 license. However, there are multiple versions of this license family and 'bsd-style' could refer to all of them. This problem also occurred when the license relevant text only contained links to a website. Overall, this issue was found in cases 1, 2, 10, 12, 14, 15, 16 and 19.

- Scancode: The crawler actually guessed the correct answer in the specific case mentioned above. It conclude that 'bsd-style'-license refers to the BSD-3.0 license. However, the crawler did not check the corresponding license file. Thus, it merely made a correct guess. This handling could potentially prove to be error-prone.
- FOSSology: The crawler did not give a definitive answer in the cases we inspected. It merely marked the file with a 'LicenseRef-BSD' tag. This is supposed to serve as a prompt for the user to manually check the file for the licensing text.

**2. Context**

This error was mostly found while scanning larger files that contain text unrelated to the actual license the project is published under. The crawlers scan these text files for mentions of any license. However, this text does not necessarily refer to actual licensing information related to the project. Case 19 for example looks at an FAQ file in which a question asks if a component is published under the 'GNU General Public License'. The actual answer to this question states that it is not. In order to come to the right conclusion, a crawler must be able to recognize this. Overall, this issue was found in cases 6, 19, 21 and 24.

- Scancode: The crawler marked these cases with an actual license hit.
- Fossology: The crawler marked these files with LicenseRef-tags. Prompting the user to look at the corresponding files.

**3. Incorrect Version**

In some of the cases, the crawler was unable to determine the exact version of a specific license family, even if sufficient information was available. Excluding the

cases from category 1.,this issue was present in case 11, 23, 24 and 25.

- Scancode: The crawler identified the correct version in 2 of the cases.
- FOSSology: The crawler made a mistaken in 3 of the 4 cases.

**4. False evaluation**

In several of the cases checked, we were not able to understand why a license crawler failed. This includes crawlers marking licenses that were not actually present, as well as failing to find the correct license. Overall, this issue was found in cases 3,4,7,8,9,13.

Looking at all the conflict situations we investigated, we found out that Scancode oftentimes assumes the license based on insufficient information. FOSSology on the other hand is more conservative with its statements, by only marking the files with a Reference-tag. Thus, considering the time spent scanning the projects, our evaluation of the conflict situations and the handling of the error categories, we found Fossology to be the faster more reliable tool.

## 2.7 Limitations

Firstly, one of the issues we faced while conducting our benchmark is the sheer volume of data. All sampled projects together have a total of 292573 files. As a result, it was not possible to check every single file by hand and the random selection of conflict situations we made could potentially not be representative of the actual outcome.

Secondly, a lot of the conflicts repeated themselves many times across the same project. Some projects copied the licensing header into every file of a directory and it's subdirectories. This resulted in thousands of conflicts being virtually the same. This problem can be seen in table 2.1-2.3, as many conflicts share the same pattern.

Thirdly, we found that the documentation of the crawlers workflow was often insufficient. The developers did not properly document every step of their algorithms making the detailing of the used algorithm difficult.

Lastly, it is important to consider that the sampled crawler are still in development. The projects are popular open source projects and each have a community working on them. Thus, potential changes in the future might improve the results of the crawlers. This makes disregarding any of the software tools for further study problematic.

## 2.8 Future work

We suggest that there should be further research done investigating the identified error categories. As the two crawler agreed on a large amount of files, these conflict situations provide valuable insight. An attempt could be made to add a solution to resolving the references found in some of the files by following links or looking for the files mentioned in the header. Furthermore, the problem of understanding context should be addressed. Some sort of algorithm is necessary to recognize keywords outside of licensing information. Furthermore, as we were only able to look at 25 conflict situations due to time constraints, a more thorough approach could uncover additional information.

Another issue that could arise through the context error category is licensing trolls. A potential project could maliciously add a simple 'not licensed under' to their header. Most likely none of the investigated crawlers would be able to recognize that the statement means the opposite of a hit. Thus, a prevention algorithm should be considered.

## 2.9 Conclusion

In this thesis we took an in-depth look at the state of the art of open source license crawler. By creating a sampled test set of open source projects, we were able to benchmark 6 of the existing license crawlers in a direct competition.

This showed us that 2 of the 6 software tools strongly outperformed the rest. Overall, Scancode and FOSSology found 83.24% of all licenses in our benchmark. Thus, we decided to look at these software tools on a directory tree level. By considering conflict situations in the output, we were able to find out that both crawlers have only few complete misses. However, Scancode has a much higher rate of false positives, while FOSSology is more conservative in it's evaluation. Furthermore, we were able to classify the errors made by the crawlers into 4 different categories. These categories represent difficult situations for license crawlers to evaluate.

For future research we suggest to expand the scope of the conflict analysis. Additionally, more research is necessary to find solutions to the mentioned error categories.

# Appendix A Sampling information

Used projects:

| | | |
|---|---|---|
| AFNetworking-master | incubator-echarts-master | react-native-master |
| angular-master | jekyll-master | react-router-master |
| async-master | jQuery-File-Upload-master | redis-5.0 |
| atom-master | julia-master | redux-master |
| axios-master | keras-master | requests-master |
| babel-master | kotlin-master | Rocket.Chat-master |
| bitcoin-master | laravel-master | rust-master |
| bootstrap-master | lodash-master | RxJava-2.x |
| brackets-master | lottie-android-master | scikit-learn-master |
| caddy-master | mermaid-master | SDWebImage-master |
| cpython-master | meteor-master | select2-master |
| d3-master | moment-master | serverless-master |
| discourse-master | node-master | shadowsocks-windows-master |
| express-master | normalize.css-master | slate-master |
| flask-master | nvm-master | socket.io-master |
| fullPage.js-master | nylas-mail-master | spring-boot-master |
| Ghost-master | oh-my-zsh-master | swift-master |
| gogs-master | parcel-master | tensorflow-master |
| grafana-master | pdf.js-master | three.js-master |
| gulp-master | pixi.js-master | vscode-master |
| hexo-master | preact-master | vue-master |
| html5-boilerplate-master | prettier-master | x64dbg-development |
| httpie-master | prometheus-master | yarn-master |
| hugo-master | quill-develop | you-get-develop |
| immutable-js-master | rails-master | zxing-master |

# Appendix B Conflict situations

angular-master/packages/animations/browser/src/render/css_keyframes/direct_style_player.ts

angular-master/modules/benchmarks/e2e_test/tree_spec.ts

bitcoin-master/src/qt/transactionfilterproxy.h

bitcoin-master/src/test/cuckoocache_tests.cpp

brackets-master/src/extensions/default/JavaScriptQuickEdit/unittest-files/jquery-ui/ui/jquery.ui.tooltip.js

brackets-master/src/nls/id/strings.js

cpython-master/Lib/platform.py

cpython-master/Lib/unittest/__init__.py

kotlin-master/core/script.runtime/src/kotlin/script/templates/annotations_deprecated.kt

kotlin-master/js/js.ast/src/org/jetbrains/kotlin/js/backend/ast/JsBreak.java

node-master/deps/v8/test/message/fail/rest-param-object-setter-sloppy.js

node-master/deps/v8/test/mjsunit/harmony/regexp-property-lu-ui3.js

node-master/deps/v8/test/cctest/gay-precision.cc

node-master/deps/v8/tools/unittests/testdata/testroot2/test/sweet/testcfg.py

node-master/deps/v8/src/string-hasher.h

node-master/deps/v8/src/compiler/type-narrowing-reducer.cc

node-master/deps/icu-small/source/i18n/simpletz.cpp

node-master/deps/icu-small/source/common/ubidiln.cpp

node-master/deps/zlib/FAQ

pdf.js-master/test/resources/reftest-analyzer.js

prometheus-master/vendor/google.golang.org/api/CONTRIBUTORS

Rocket.Chat-master/packages/rocketchat-ui/client/lib/Modernizr.js

scikit-learn-master/sklearn/utils/multiclass.py

swift-master/stdlib/public/core/SipHash.swift

tensorflow-master/tensorflow/compiler/xrt/BUILD

# Literaturverzeichnis

German, D. M., Manabe, Y. & Inoue, K. (2010). A Sentence-matching Method for Automatic License Identification of Source Code Files. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering* (S. 437–446). ASE '10. Antwerp, Belgium: ACM.

Lerner, J. & Tirole, J. (2005). The Scope of Open Source Licensing. *Journal of Law, Economics, & Organization*, *21*(1), 20–56.

Rosen, L. (2005). *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall.

Stewart, K. J., Ammeter, A. P. & Maruping, L. M. (2006). Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects. *Information Systems Research*, *17*(2), 126–144.

Stol, K.-J. & Fitzgerald, B. (2018). The ABC of Software Engineering Research. *ACM Trans. Softw. Eng. Methodol. 27*(3), 11:1–11:51.

Vendome, C., Bavota, G., Penta, M. D., Linares-Vásquez, M., German, D. & Poshyvanyk, D. (2017). License usage and changes: a large-scale study on gitHub. *Empirical Software Engineering*, *22*(3), 1537–1577.