Empirical assessment of the performance of Muzeel

Christos Petalotis
2739394
VU Amsterdam
c.petalotis@student.vu.nl

ABSTRACT

Context. As developers tend to import more and more third-party packages in their code in an attempt to make their applications appealing and highly interactive for their users, the amount of unused code in websites increases as well. For this reason, it is important to optimise a web page's source code by removing any unused code from it. Lacuna and Muzeel are two tools that intend to provide a solution to this issue by detecting and removing JavaScript functions that are unused. Previous work has focused on assessing and comparing these tools on performance metrics such as PageLoadTime and the CPU bandwidth used.

Goal. The focus of this paper is on investigating the performance of Muzeel and Lacuna in terms of correctness, completeness, and precision. A comparison of the results of each tool is performed as well. The results of this study could help web developers understand whether dead code elimination tools produce satisfying results and which tool performs better in that realm.

Method. The performance of Muzeel and Lacuna was tested on 29 subjects, stemming from the TodoMVC project. The ground truth for these applications is generated using the dynamic-deadfunction-detector tool. The results of each tool were compared to the ground truth values to derive the values for precision, recall, and f-score to measure correctness, completeness, and accuracy accordingly.

Results. Muzeel's precision is measured at 63%, with Lacuna having a value of 74% for the same metric. In terms of completeness, (measured by recall) Muzeel and Lacuna have values of 68% and 64% respectively. Regarding accuracy, measured in terms of f-score, which combines precision and recall, the measure for Muzeel was 60% and for Lacuna 67%. Muzeel tends to perform better on a greater number of subjects across all metrics. However, Muzeel's advantage on these occasions is not significant. Conversely, when Lacuna outperforms Muzeel on a certain subject, the difference is considerable.

Conclusions. Lacuna's ability to detect dead JavaScript code in web applications is better than Muzeel's, despite the fact that Muzeel is able to collect more of the dead functions present in a website's source code. In both cases, the results are satisfying, suggesting that their usage can lead to the detection of a great amount of unused code in JavaScript files. However, Lacuna is preferred due to certain Muzeel limitations that were identified during this study.

Research Project 2022/2023, January, 2023, Amsterdam, The Netherlands

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Christos Petalotis. 2020. Empirical assessment of the performance of Muzeel . In *Research Project 2022/2023 - Vrije Universiteit Amsterdam, January, 2023, Amsterdam (The Netherlands).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In an attempt to make pages highly interactive and appealing to users, developers of websites introduce third-party JavaScript (JS) libraries that offer various methods to achieve these objectives. Additionally, importing existing code can speed up the development process, as various functionalities need not be created from scratch. However, using external dependencies introduces superfluous code, as the additional functionality is only partially leveraged, both during a page load and the interaction of the user with the page. It has been found that top-ranked websites contain an average of 31% of excessive code [1]. The unused code, also referred to as *dead code*, amplifies the size of a web page, affecting the performance of the page when running in the browser, as it requires more content to be downloaded, parsed and compiled[1].

There exist several approaches that attempt to reduce the size of JavaScript code that is contained within web pages and is loaded and parsed by client browsers, such as the one proposed by Lacuna[2]. Lacuna utilises different existing analysis techniques and their combinations to determine the unused functions included in the client-side JavaScript code of websites. Another tool that intends to tackle dead code in JavaScript files of websites is Muzeel[3]. To achieve this, Muzeel dynamically analyses JavaScript code loaded on a web browser to determine the used and unused functions that are part of a web page. It pre-processes the website's code by storing it locally and creating an Abstract Syntax Tree (AST) and a function call graph of size 1. Then, it discovers the dead code by interacting with the page using an automated browser, and finally eliminates any unused code identified. The result that both Lacuna and Muzeel generate is a version of the original JavaScript files that does not include the bodies of the functions that have been identified as unused, thus reducing the size of a website and improving its performance when served to the end user's browser [2].

The purpose of this paper is to analyse the performance of Muzeel and Lacuna in terms of correctness, completeness and accuracy across a number of JavaScript frameworks. This is done using the results provided by the dynamic-deadfunction-detector[4] on the different subjects as the ground truth. After the evaluation of each tool has been completed, a comparison of the measured metrics of

[1] https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4
[2] https://github.com/S2-group/Lacuna
[3] https://github.com/comnetsAD/Muzeel
[4] https://github.com/Kishanjay/dynamic-deadfunction-detector

them is conducted. The results of this study are intended to provide developers with insights into the performance of Muzeel and Lacuna in terms of their ability to eliminate dead JavaScript code from websites developed using one of the 29 explored JavaScript frameworks.

2 RELATED WORK

This section highlights relevant scientific papers that also look at Muzeel's performance and underlines the differences between these papers and the experiment presented in the current paper, while also pointing out the way the discussed papers are complemented by the current study.

Kupoluyi conducted an experiment that measures and compares the performance of the Muzeel tool in terms of web quality metrics and resource utilisation [3]. The measures of First Contentful Paint (FCP), Speed Index (SI), Page Load Time (PLT), CPU bandwidth and battery consumption were acquired and compared for the original and the Muzeel'ed (as the authors of the paper refer to it) versions of a web page. The experiment described in this report differs as the tool's performance is assessed in terms of its ability to detect dead JavaScript functions on web application code.

Zakir et al. extend the findings of the above study and perform a comparison between Lacuna and Muzeel using the FCP, SI, and PLT web performance metrics on websites hosted on a remote server [2]. In this study, a comparison of the same tools is conducted, however, the assessment takes place on locally hosted websites, that can be controlled by the experimenter. Additionally, this paper attempts to compare Lacuna and Muzeel on performance metrics that evaluate the correctness, completeness, and accuracy of the tools, instead of the web metrics discussed in [2].

3 EXPERIMENT DEFINITION

Following the GQM framework [4], the purpose and scope of the experiment described in this paper are defined, along with the research questions that are intended to be answered and the metrics used to achieve this. The objective of the experiment is defined formally in the following manner:

"Analyse and compare Muzeel and Lacuna for the purpose of evaluating their performance with respect to their ability to identify JavaScript dead functions code from the point of view of web developers in the context of JavaScript web applications".

A visual representation of this definition is shown in Figure 1.

The experiment goal is refined into the following research questions, considering each of its aspects:

RQ1: What is the *performance* of *Muzeel* in terms of detecting dead JavaScript functions?

RQ2: What is the *performance* of *Lacuna* in terms of detecting dead JavaScript functions?

RQ3: How is the performance of *Muzeel* compared to the *performance* of *Lacuna* in the task of detecting and eliminating dead JavaScript functions?

The evaluation of Muzeel's and Lacuna's performance to identify dead JavaScript functions is assessed across three sections: correctness, completeness, and accuracy.

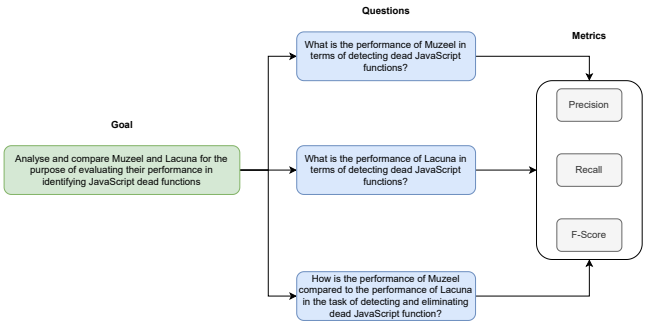

Figure 1: Visual representation of the GQM

The correctness of the results for Muzeel and Lacuna is evaluated using the precision metric. This metric allows us to understand how relevant are the results that are produced, by showing the percentage of functions that are correctly identified as dead, considering the collection of functions that are marked as unused by Muzeel. Low levels of correctness could potentially introduce errors and unwanted behaviour, due to the absence of JavaScript functions that are required for the website to operate appropriately.

Recall is used to measure the completeness of the results Muzeel and Lacuna produce. This metric reveals the ability of the tool to identify relevant results, meaning functions that are dead. In the case of low levels of completeness, employing Muzeel or Lacuna may lead to unnecessary overhead, which does not generate results that are useful and can improve a web page's performance when served on a web browser.

The accuracy of the results Muzeel and Lacuna generate is gauged using the f-score metric. The value of this metric is a measurement of both the completeness and the correctness of the tools and is defined as the harmonic mean of precision and recall[5].

The performance metrics that are measured for Muzeel are compared with the corresponding metrics for Lacuna in order to realise Muzeel's competition and determine which one of the two tools demonstrates a better ability to identify functions that are not used by a certain website. For this reason, 29 subjects are used, each one of them corresponding to a different JavaScript framework, with which the same application has been developed. In this way, we realise the performance of Muzeel across a wide range of developing options for JavaScript-based web pages.

The target audience of this paper is web developers who develop websites with user satisfaction in mind. In their attempt to make a web page as appealing and as functional as possible, developers often introduce unused code through third-party libraries, which can negatively affect the performance of their products. This is why their work could be greatly benefited from software that analyses all the dependencies of their product and indicates or eliminates dead code. Thus, the findings of this research could provide web developers with an indication of the ability of dead code detection tools to correctly recognise functions that are not used, as well as suggest which kind of tool to use depending on the JavaScript framework that is used to develop a website.

[5]https://www.educative.io/answers/what-is-the-f1-score

4 EXPERIMENT PLANNING

The objective of our experiment is to investigate and compare the performance of Muzeel and Lacuna on the task of detecting dead JavaScript code. To achieve that, it is important to define the context and planning of the research in a way that allows for clear and objective results to be derived. For this, we follow the guidelines proposed by Wohin et al. [5]

4.1 Subjects Selection

The ability of Muzeel and Lacuna to properly identify and eliminate dead code from web applications is evaluated on a collection of 29 websites. These applications originate from the TodoMCV project[6], a compilation of the same application, developed in various JavaScript frameworks. This application is a to-do list manager providing the user with the ability to add an item to the list, remove an item from the list, mark items as completed, and change views between "All", "Active" and "Completed" list items.

The selection of subjects is based on the subjects that were used in the evaluation of Lacuna in [6] (referred to as the original Lacuna evaluation study). In this paper, 39 subjects are used. However, in the current study, Lacuna failed to generate any results for 10 of these web applications, namely the ones developed using the following JS frameworks: ariatemplates, aurelia, closure, exoskeleton, knockoutjs, polymer, puremvc, rappidjs, riotjs, vanilla-es6.

Table 1 lists the web applications that are used as subjects in this study, along with information on the number of total, dead and alive JavaScript functions for each of them. These values, which are retrieved using the dynamic-deadfunction-detector tool, constitute the ground truth for our study and will be used as a reference point when calculating the precision, recall and f-score for Muzeel and Lacuna. The framework used to develop each version of the TodoMVC application is used as an identifier for it.

4.2 Experimental Variables

In this study, we explore the performance of two dead code elimination tools on websites. The structure and functionality of each tool presumably result in different outcomes for each one, affecting the final values the metrics for performance assessment are assigned. Therefore, the type of tool is considered to be the independent variable in this experiment. In particular, the factor considered to answer RQ1 is the Muzeel tool, while the factor for RQ2 is the Lacuna tool. RQ3 can be answered by comparing the results that each tool produces; thus, both are considered independent variables in this case.

Additionally, each aspect of the evaluation of Lacuna and Muzeel's performance, as discussed in Section 3, introduces one dependent variable. In particular, to measure the correctness of Muzeel and Lacuna (RQ1), the dependent variable of precision (P) is used. To calculate precision, the number of correctly identified and wrongly identified dead functions is used as follows:

$$P = \frac{TP}{TP + FP}$$

[6]https://todomvc.com/

Table 1: List of web applications used in the experiment and relevant information

Subject	#Functions	#Alive Functions	#Dead Functions
ampersand	774	390	384
angularjs_require	127	80	47
backbone	950	408	542
backbone_require	127	88	39
canjs	1105	459	646
canjs_require	128	76	52
dijon	834	299	535
dojo	1074	596	478
enyo_backbone	20	15	5
flight	127	88	39
jquery	869	386	483
knockoutjs_require	128	76	52
lavaca_require	981	304	677
mithril	136	78	58
olives	533	232	301
ractive	932	541	391
react	1848	948	900
react-alt	2033	1052	981
sapui5	16	12	4
serenadejs	400	165	235
somajs	342	172	170
somajs_require	128	80	48
spine	999	456	543
troopjs_require	130	79	51
typescript-angular	1440	674	766
typescript-backbone	1243	459	784
typescript-react	1105	696	409
vanillajs	131	116	15
vue	622	352	270

To answer RQ2 (completeness) the dependent variable computed is the recall (R) metric, which depends on the number of TP and FN:

$$R = \frac{TP}{TP + FN}$$

Finally, the assessment of Muzeel and Lacuna in terms of accuracy is made considering the f-score (F) of each tool for the various subjects. This metric constitutes the harmonic mean of precision and recall and is defined as follows:

$$F = 2 \times \frac{P \times R}{P + R}$$

To measure the performance of Muzeel and Lacuna, the number of true positives (TP), false positives (FP), and false negatives (FN) are utilised. More specifically, a result is considered true positive in the case that Muzeel or Lacuna considers a function as dead, and it is denoted as dead in the ground truth, too. Contrastingly, a false positive means that Muzeel or Lacuna identified a function as dead, while the ground truth considers the same function to be alive. A false negative refers to a function that is recognised as alive by Muzeel or Lacuna, but it is actually unused according to the ground truth. The values of P, R and F range in [0, 1], with higher values for each metric indicating a better ability of Muzeel or Lacuna to identify JavaScript functions and distinguish which ones are unused and which ones are alive.

4.3 Experiment Design

In order to measure P, R, and F for each dead code detection tool, we apply Muzeel and Lacuna to each of the subjects. The subjects are hosted on the author's computer and accessed through localhost by the tools. The performance results depend only on the ability of each tool to properly identify dead functions in JavaScript code and cannot be affected by the order of execution of the subjects or any other factor, such as network or time to complete execution. For this reason, the order in which the subjects are executed is not important. Each tool is applied twice on each subject to verify that the generated results are consistent across runs.

4.4 Data Analysis

After each tool, Muzeel and Lacuna, has been applied to the list of subjects listed in table 1, and the number of alive and dead functions in each case has been derived, the analysis of each tool can be performed. For this reason, a list of descriptive statistics is calculated for the performance of each tool in terms of correctness, completeness, and accuracy. Additionally, the use of box plots for these aspects helps visualise the differences in performance between Muzeel and Lacuna. This way the distribution of the data is realised and the research questions defined in this paper can be answered.

5 EXPERIMENT EXECUTION

In this section, the preparation and execution of the analysed tools are described, along with the hardware and software specifications on which these tools were run[7]

5.1 Preparation and Execution

5.1.1 ***Ground Truth.*** Prior to evaluating the correctness, completeness, and accuracy of the tool, it is first necessary to establish the true set of dead and alive functions (i.e., the ground truth). This ground truth is derived from the pre-established experimental design used to evaluate Lacuna [X]. The tool operates by instrumenting each function of a web application with a logging statement, which records whether the function is executed or not at run-time. To facilitate the recording of function executions, the tool utilizes the pre-existing test suite available within the TodoMVC project. The TodoMVC project provides a set of web applications, each with its own test suite, which exercises all functionalities of the respective application. The test suite is then utilised to run each web application automatically, allowing the script to record the executed functions. As a result, functions marked as executed are considered alive, while those that have not been executed are deemed dead. The number of alive and dead functions for each web application is presented in Table 1.

It is important to highlight that the dynamic-deadfunction-detector [8] has not been updated for the past 3-4 years. As a result, the node packages and scripts had to be adapted to function correctly with the current hardware. Further details regarding the hardware and software specifications can be found in Section 5.2, and the updated application version can be accessed on GitHub[9].

5.1.2 ***Muzeel.*** The execution of Muzeel on the subjects of this experiment was carried out according to the steps detailed in the README file of the tool's GitHub repository[10]. More specifically, a MySQL database must be created that follows the SQL structure defined in the *templateDB.sql* file (it can be imported into the database to replicate the necessary structure). The details of the database need to be updated in the relevant sections in the source code of Muzeel, before running the appropriate combinations of proxy servers and Python scripts to cache and collect results regarding the dead and alive functions of the desired endpoints. Due to a limitation in Muzeel, only one endpoint can be tested at a time. Muzeel is not able to iterate over items from a list of target endpoints. For this reason, a custom Python script was developed, which executes Muzeel on each target endpoint. For this reason, an additional list of endpoints, different from the prebuilt one, needs to be constructed, which holds all the endpoints on which we want to operate Muzeel. This script requires the cache and proxy servers to be running in the background, throughout its execution. A detailed walk-through of the steps followed to run Muzeel on the collection of subjects for this experiment can be found on the corresponding GitHub repository[11].

It is important to mention that Muzeel's source code was modified to accommodate sites hosted on localhost. This was necessary as the latest version of the tool works as expected only on web pages hosted on external servers, while not producing valuable results for localhost. More specifically, the proxy settings for the Selenium instances that are used to cache and navigate a web page were modified as needed.

5.1.3 ***Lacuna.*** In the original GitHub repository on the evaluation of Lacuna[12], certain steps are outlined regarding the execution of Lacuna and the generation of results on dead and alive functions. These steps were followed for the setup of Lacuna on the collection of selected subjects. The step regarding the execution of Lacuna on the subjects was performed using a custom Python script. Furthermore, the derivation of statistics on its performance in terms of correctness, completeness, and accuracy was performed with another custom Python script.

In particular, initially, the tested web applications are normalised, by removing any inline scripts that are present and locally hosting

[7] Sections 5.1, 5.2 and 5.3 were conducted in collaboration with Luka Krumpak

[8] https://github.com/Kishanjay/dynamic-deadfunction-detector
[9] https://github.com/lkrumpak/lacuna-evaluation-ground-truth
[10] https://github.com/comnetsAD/Muzeel
[11] https://github.com/ChrisPetalotis/muzeel_evaluation
[12] https://github.com/S2-group/Lacuna-evaluation

any externally hosted scripts. This was done using the appropriate normaliser script, as described in the README file contained within the relevant GitHub repository for the performance evaluation of Lacuna[13]. After that, our custom script is run to get the Lacuna results about dead and alive functions. For this purpose, Lacuna was employed on each subject using either the combination of the TAJS and the dynamic analysers or just the dynamic analyser, to reduce the time it takes to generate results, rather than using every available analyser to Lacuna. This is done because these settings lead to the best performance results for Lacuna [6]. Once the Lacuna results were generated, the developed Python script that induces performance metric values and statistics about them is run, populating the values of precision, recall, and f-score for each subject, and the tool in general across all subjects.

5.2 Setup

The laptop used to run the experiment was a Macbook Pro, equipped with an Apple M1 Pro processor, 16GB of RAM and operated on the Ventura 13.0.1 operating system. Furthermore, the experiment was executed using Node version 18.12.1 and npm version 8.19.2, while automated scripts to evaluate the tools were executed using Python version 3.10.9.

5.3 Analysis

All analysis is conducted using the Python language version 3.10.9. The data analysis and manipulation module *pandas* is used for data exploration, analysis, and plotting.

6 RESULTS

In this section, certain statistics on the precision, recall, and accuracy of each tool's ability to detect dead functions in JavaScript code are presented. These statistics are based on the performance of Muzeel and Lacuna across all subjects considered in this experiment. Additionally, the values of these metrics for each tool and each subject are provided to allow for a more detailed comparison of the performance of Muzeel and Lacuna on each JavaScript framework separately.

In Table 2 statistics regarding the central tendency and dispersion of the correctness of Muzeel and Lacuna are listed. Table 3 presents the same statistics related to Lacuna's and Muzeel's ability to identify relevant results (i.e. completeness in terms of recall). In Table 4 descriptive statistics on the accuracy of these tools are displayed in terms of f-score.

The average performance of Lacuna in detecting dead functions was found to be better than Muzeel in terms of precision and f-score, using the mean values. However, when the median values are considered, Muzeel performs better in terms of completeness (i.e. recall) and accuracy (i.e. f-score). It is interesting that when the f-score is considered, Muzeel gets a higher value for the median, but a lower value for the mean, compared to Lacuna. More specifically, the mean and median values for accuracy for Lacuna are 0.67 and 0.70 respectively, while the same values for Muzeel are 0.60 and 0.73. This could mean that the f-score measures for Muzeel tend to have higher values than Lacuna, but with a few very low values that result in a low mean f-score value when compared to Lacuna.

[13]https://github.com/lkrumpak/lacuna-evaluation-ground-truth

Table 2: Statistics on the precision of Muzeel and Lacuna

Precision Statistics	Lacuna	Muzeel
Min.	0.14	0
Max.	0.99	1
Median	0.80	0.79
Mean	0.74	0.66
SD	0.22	0.34
CV	29.85	51.63

Table 3: Statistics on the recall of Muzeel and Lacuna

Recall Statistics	Lacuna	Muzeel
Min.	0.2	0.08
Max.	0.87	1
Median	0.72	0.75
Mean	0.64	0.69
SD	0.17	0.20
CV	26.83	29.52

Table 4: Statistics on the f-score of Muzeel and Lacuna

F-Score Statistics	Lacuna	Muzeel
Min.	0.2	0
Max.	0.91	0.92
Median	0.70	0.73
Mean	0.67	0.60
SD	0.18	0.30
CV	26.92	49.52

With respect to recall, i.e., completeness, Muzeel performed better in both median and mean values. Moreover, it can be observed that Lacuna's performance has less dispersion consistently compared to Muzeel across all performance metrics, as shown by the standard deviation (SD) and the coefficient of variation (CV). Especially in the case of CV, the difference is significant for accuracy and correctness.

Furthermore, we can see that the minimum performance values produced by Lacuna are consistently higher than those of Muzeel at an average of 12%. In contrast, the maximum values are always higher in the case of Muzeel, with the largest difference observed for recall, where the difference is 13%.

The distribution of the precision, recall, and f-score values for Muzeel and Lacuna on the collection of subjects are depicted as boxplots in graphs 2, 3, and 4. These box plots help to visualise the descriptive statistics presented in tables 2, 3, and 4 and emphasise the large dispersion that characterises the results of Muzeel.

In Table 5 the performance metrics for Lacuna and Muzeel on each distinct subject are presented. It can be observed that, in most cases, the precision and recall values are close for both tools, resulting in similar f-score values. Counting the times each tool exhibits

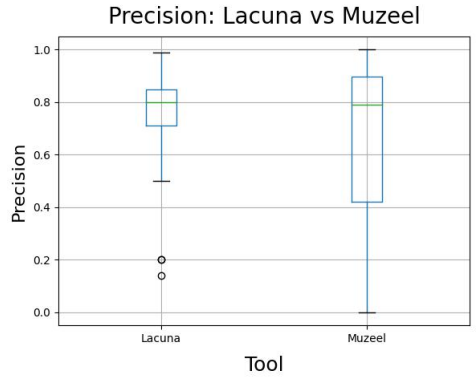

Figure 2: Distribution of precision for Lacuna and Muzeel

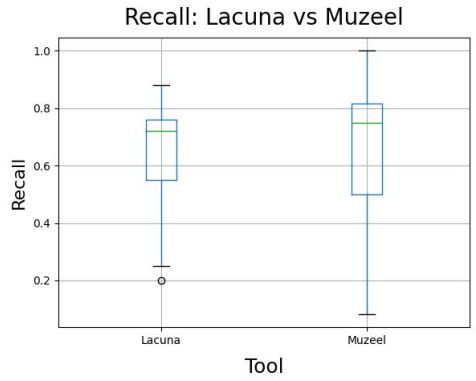

Figure 3: Distribution of recall for Lacuna and Muzeel

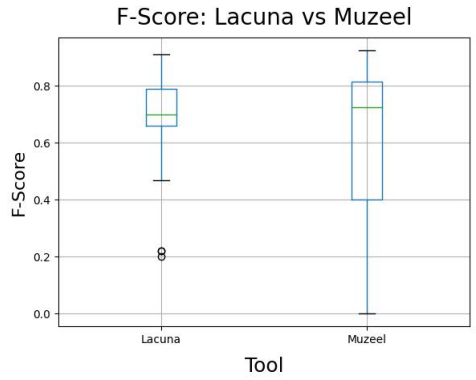

Figure 4: Distribution of f-score for Lacuna and Muzeel

better performance, we notice that the number of times that Lacuna has better performance than Muzeel is 11 concerning precision, 10 concerning recall and 13 for f-score. On the other hand, Muzeel performs better than Lacuna 18 in terms of precision, 19 times in terms of recall, and 16 out of 29 times in terms of f-score. We observe that Muzeel tends to perform better on a greater number of subjects across all collected metrics.

However, when Lacuna generates better precision results than Muzeel, the values are higher by 35% on average, while Muzeel performs better than Lacuna by 8% on average, in the occasions it has greater precision measurements. Similarly, for the f-score, when Lacuna outperforms Muzeel, it does so by 22% on average, while this number for the instances in which Muzeel outperforms Lacuna is only 6%. The average gain of each tool compared to the other in terms of recall is 10% for Lacuna and almost 12% for Muzeel. It can be observed that even though Muzeel performs better than Lacuna on more occasions, on average the better performance is not significant. On the other hand, the number of times Lacuna performs better than Muzeel is smaller; however, when this happens, the difference is considerable. As a result, the average performance of Lacuna is better in terms of correctness and accuracy, as described above and presented in Tables 2 and 4.

Furthermore, the subjects with identifiers enyo_backbone and sapui5 are of particular interest. In these cases, Muzeel was unable to correctly identify the total of the dead functions in the application code, while Lacuna could correctly detect unused functions at a rate of 20%. However, Muzeel's completeness was excellent in these cases, managing to detect every dead function in the subjects' code, while Lacuna's performance stayed close to 20% for this metric too. This behaviour and unusual performance measures for both Lacuna and Muzeel might be explained by the file structure of the Sapui5, Enyo and Backbone.js JavaScript frameworks and can be the subject of future work. Additionally, a great difference in performance values can be noticed for the backbone_require subject, which combined the Backbone.js and RequireJS frameworks. In this case, Lacuna performs better than Muzeel by 30-50% across all metrics.

7 DISCUSSION

This section interprets the results presented in Section 6 and explains their implications for web developers, with respect to each research question.

RQ1:*What is the performance of Muzeel in terms of detecting dead JavaScript functions?*

The performance of Muzeel has been assessed across three different aspects: correctness, completeness, and accuracy corresponding to the precision, recall, and f-score metrics respectively. It was found that Muzeel is able to correctly detect around 66% of dead functions in JavaScript code on average using the mean, and 79% when the median is considered. At the same time, it is able to identify around 69% and 75% of all the dead functions in web applications, considering the mean and median accordingly. As a result, the performance of Muzeel in terms of accuracy is 60% on mean and 73% on median. We observe that the mean values are always lower than the median ones, indicating that there are more extreme values in the bottom 50% than in the top 50%.

Taking the performance measures into account, we can infer that Muzeel's performance in detecting unused JavaScript functions is relatively high. However, using only this tool, a large chunk of functions would not be identified, thus its use should be complemented by another tool or manual code inspection. This conclusion is reinforced by the fact that the level of dispersion around the

Table 5: Performance metric of Muzeel and Lacuna for each subject

Subject	Precision Lacuna	Precision Muzeel	Recall Lacuna	Recall Muzeel	F-Score Lacuna	F-Score Muzeel
ampersand	0.78	0.79	0.75	0.76	0.76	0.78
angularjs_require	0.84	0.06	0.57	0.49	0.68	0.11
backbone	0.74	0.78	0.74	0.75	0.74	0.76
backbone_require	0.85	0.35	0.72	0.46	0.78	0.40
canjs	0.99	1.00	0.75	0.75	0.86	0.86
canjs_require	0.86	0.05	0.58	0.50	0.69	0.09
dijon	0.99	1.00	0.76	0.75	0.86	0.86
dojo	0.80	0.77	0.69	0.67	0.74	0.71
enyo_backbone	0.20	0.00	0.20	1.00	0.20	0.01
flight	0.79	1.00	0.69	0.67	0.74	0.80
jquery	0.84	0.86	0.74	0.75	0.79	0.80
knockoutjs_require	0.86	0.83	0.58	0.48	0.69	0.61
lavaca_require	0.98	0.42	0.36	0.08	0.52	0.14
mithril	0.50	0.61	0.45	0.53	0.47	0.57
olives	0.78	0.81	0.48	0.49	0.59	0.61
ractive	0.75	0.71	0.83	0.83	0.79	0.77
react	0.62	0.90	0.82	0.91	0.70	0.90
react-alt	0.62	0.91	0.80	0.89	0.70	0.90
sapui5	0.20	0.00	0.25	1.00	0.22	0.00
serenadejs	0.98	1.00	0.73	0.75	0.84	0.86
somajs	0.97	1.00	0.85	0.86	0.91	0.92
somajs_require	0.85	1.00	0.58	0.50	0.69	0.67
spine	0.67	0.71	0.73	0.75	0.70	0.73
troopjs_require	0.82	0.15	0.55	0.49	0.66	0.23
typescript-angular	0.85	0.85	0.49	0.49	0.62	0.62
typescript-backbone	0.81	0.84	0.77	0.78	0.79	0.81
typescript-react	0.71	0.76	0.88	0.88	0.79	0.82
vanillajs	0.14	0.21	0.53	0.80	0.22	0.33
vue	0.80	0.89	0.79	0.82	0.79	0.85

mean is high for each metric, indicating a behaviour that is not consistent across subjects.

RQ2:*What is the performance of Lacuna in terms of detecting dead JavaScript functions?*

Regarding Lacuna, its ability to correctly identify dead JavaScript functions is measured to be around 74§% (mean) and 80% (median). Moreover, it is capable of detecting 64% of all unused functions in JavaScript code, when the mean is considered, and 72% when the median is considered. In terms of the accuracy of detected functions, which balances correctness and completeness, Lacuna's performance is measured using the f-score metric and is around 67% on mean and around 70% considering the median. The dispersion that is observed for Lacuna is low but still significant, leading to results that might not be consistent across different subjects. Furthermore, similarly to Muzeel, the performance of Lacuna may not be enough and complementary techniques could prove useful.

It is important to note that the performance metric values computed in this experiment for Lacuna are inferior to the results that the developers of the tool have collected and discussed in [6], where they evaluate Lacuna on the same metrics. As certain difficulties were faced during the implementation of Lacuna on the set of subjects as part of this study, the results may be different because of this reason.

RQ3:*How is the performance of Muzeel compared to the performance of Lacuna on the task of detecting and eliminating dead JavaScript functions?*

Based on our empirical results, Lacuna tends to identify more unused JavaScript functions compared to Muzeel in general. This behaviour is also more consistent across subjects - and consequently JS frameworks - for Lacuna, as its dispersion statistics are measured to be significantly lower than the ones for Muzeel across all subjects. This is in spite of the fact that Muzeel outperforms Lacuna in more subjects than the opposite.

Lacuna's better performance is justified by the measurements for precision (i.e. correctness) and f-score (i.e. accuracy). However, as discussed in 6, Muzeel performed better in terms of completeness. This means that Muzeel is able to identify a higher fraction of the dead JavaScript function in web apps than Lacuna, despite performing worse in terms of precision due to a large number of FPs by Muzeel. A possible explanation of this could lie in the fact that

Muzeel handles each function independently, "regardless of whether it is nested in another function or not" [3]. Thus, functions that are not considered unused by Lacuna, and the dynamic-deadfunction-detector when constructing the ground truth, could be detected by Muzeel, leading to the observed performance difference in completeness and the high number of False Positives by Muzeel. This means that Muzeel tends to detect more functions than have been considered dead in the ground truth, along with the correct dead functions. Thus, even though the number of TP is high and the number of FN is low - resulting in high recall values - the number of FP tends to soar in certain cases for Muzeel, causing the correctness (precision) metric to get low values. This behaviour of Muzeel could become problematic and affect the functionality of a website. If functions that are essential for the use of a web application are marked as dead and their contents are eliminated, its navigation and usage might not be possible anymore. It should be noted that Lacuna can be configured with an option which allows it to replace the contents of a dead function with a synchronous call of its original code. Therefore, even if Lacuna gets low values for precision, the functionality of the web application is not lost, regardless of the performance hit the synchronous call to the function's code introduces.

One limitation that Lacuna has is that its application may require a considerable amount of time, up to 30 minutes, to analyse the source code of a web application and to detect dead JavaScript functions when the TAJS analyser is involved. Since this analyser is the most effective one, in combination with the dynamic analyser, it is expected to be used often by developers when Lacuna is used. Therefore, developers should be aware of this fact and consider the lengthy period that is needed when attempting to eliminate dead code from JavaScript files using Lacuna. That said, using only the dynamic analyser is very fast and produces similarly good results. Muzeel's application on a subject is very fast as well, as it dynamically scans the JavaScript files for dead functions.

In both cases, the tool's ability to detect dead code leaves a lot of space for improvement and could result in websites with a lot of dead code remaining after the tools are run, according to the data collected in this study. This is an indication that both tools are potentially not good enough on their own and their use should be complemented by other dead code elimination techniques too if one wants to achieve complete elimination of dead code in their websites. Additionally, Muzeel should be used with caution, particularly for websites developed using either the Sapui5 or a combination of the Enyo and Backbone.js frameworks, as essential functionality might get removed, with no way to reproduce the original service. That said, their usage can be clearly justified, as a large percentage of dead functions are expected to be detected and eliminated, making their adoption a worthwhile task for developers. Combined with extensive testing of the functionality of a web application after one of the tools has run, the benefits are clear. Lacuna might be preferred due to the discussed limitations that Muzeel has, the considerable difference between Lacuna and Muzeel when Lacuna operates better, and the insignificant advantage that Muzeel shows when it performs better than Lacuna.

8 THREATS TO VALIDITY

In this section, possible threats to the validity of the research are discussed to evaluate the generalisability and soundness of the experiment results. This is done using the classification provided by Cook and Campbell (1972) [7].

8.1 Internal Validity

This category comprises threats strongly related to the design and execution of the experiment.

8.1.1 ***Selection.*** The subjects examined in this paper were selected according to the group of subjects used in [6], which evaluates Lacuna on the same performance metrics explored in this paper. A requirement for their inclusion is that both Muzeel and Lacuna can be applied to the subjects.

Initially, all 39 subjects from the original sample were examined. Surprisingly, we were unable to reproduce Lacuna on 10 of these subjects, even though this was possible for the original Lacuna evaluation study. These subjects are those with labels ariatemplates, aurelia, 'closure', exoskeleton, knockoutjs, polymer, puremvc, rappidjs, riotjs, vanilla-es6. Furthermore, Lacuna failed to use the combination of TAJS and dynamic analysers for certain subjects, as the execution of TAJS resulted in errors in these cases. This was true for the web apps with labels backbone_require, somajs_require, knockoutjs_require, lavaca_require, canjs_require, and troopjs_require. For these subjects, the results were retrieved using only the dynamic analyser, revealing a potential problem of the TAJS analyser on pages developed using RequireJS. This was not the case for Muzeel, which was applied to all subjects with no problems.

The fact that we weren't able to implement Lacuna on certain subjects, led to a collection of 29 subjects being used in this study, which does not include the ones for which Lacuna failed to run.

8.1.2 ***Reliability of Measures.*** There are no environmental factors that could interfere with the results that each tool produces, as the same files are always served for a certain web application. The tools' ability to retrieve and analyse these files is not hindered by any element and depends solely on each tool's ability to do so.

That said, to ensure that the results generated by each tool are consistent and, consequently, the computed measures are reliable, each tool was executed three times on the same list of subjects. The outcome was the same in every execution cycle.

8.2 External Validity

Threats to external validity restrict the level of generalisability of the experiment results.

8.2.1 ***Interaction of selection and treatment.*** A main threat to the external validity of the experiment is whether the results examined are applicable to the real world and can be generalised to web applications not explored in this paper. This concern is enhanced by the fact that the experiment is performed on toy web applications, run on localhost, as an implementation on real-world subjects could potentially result in different conclusions about the performance of Muzeel and Lacuna. However, this is unlikely as the same files would be served in both controlled (i.e. experiment) and uncontrolled (i.e. real-world) settings by each website and picked

up by the tools, despite any environmental factors, such as network and browser selection.

The generalisability of the observed results in this study is strengthened due to the variety of frameworks, or combinations of frameworks explored. This sample of frameworks is used by a large percentage of real-world websites, thus making the experiment conclusions applicable to a large number of web applications and representative of the broader web application population.

8.3 Construct Validity

Factors that threaten the construction of the experiment constitute threats to the construct validity.

8.3.1 ***Inadequate pre-operational explication of constructs.*** Following the GQM approach, the constructs of the performed research were established early on, before the design and operation of the experiment and the collection of the measures. This way, the objective, research questions and metrics that are used to answer them are determined setting the foundation for the next steps. Additionally, the dependent and independent variables are specified in Section 4.

9 CONCLUSIONS

This study evaluates the performance of two dead code elimination tools for the JavaScript programming language, Muzeel and Lacuna, and compares them. This is done by evaluating the accuracy, completeness, and precision of the tools using the precision, recall, and f-score metrics. The experiment was conducted on 29 web applications, stemming from the TodoMVC project - a collection of to-do list managers - with each subject developed using a different JavaScript framework, or even a combination of such frameworks.

The derived results show that Lacuna has an average precision of 74%, recall of 64% and f-score of 67% for the task of identifying dead JS codes. Considering Muzeel it can correctly detect around 66% of the dead functions in JavaScript files, can recognise 69% of all the unused functions and has an accuracy of around 60%. When these two tools are compared to each other, it is found that Lacuna's performance in detecting unused JavaScript functions in web applications is superior to the one Muzeel demonstrates. Combining this fact with the limitations of Muzeel related to the potential elimination of a big chunk of alive, or used, JS functions in certain cases and frameworks leads to a suggestion in favour of Lacuna when a comparison between Muzeel and Lacuna is performed. That said, both tools can offer significant benefits by detecting and, eventually, removing a high percentage of unused JavaScript code from web application source code; thus their usage is recommended for web developers. Complementing these tools with the utilisation of additional dead code detection tools might offer even better overall performance, detecting and eliminating the entirety of the dead code in JavaScript files.

Possible future work that evaluates Muzeel and Lacuna on real-world web applications, served by remote web servers and used by a large number of people, can be conducted to realise the generalisability and persistence of the collected results in the real world too. Additionally, it would be interesting to further examine the reasons regarding the metric values computed for the Sapui5, Enyo and Backbone.js frameworks for both tools. It is possible that the subjects developed using these frameworks were not set up correctly or that the structures of these frameworks incommode the operation of dead-code elimination tools or the reason could lie elsewhere.

REFERENCES

[1] U. Goel and M. Steiner, "System to identify and elide superfluous javascript code for faster webpage loads," *arXiv preprint arXiv:2003.07396*, 2020.
[2] T. Kupoluyi, M. Chaqfeh, M. Varvello, W. Hashmi, L. Subramanian, and Y. Zaki, "Muzeel: A dynamic javascript analyzer for dead code elimination in today's web," *arXiv preprint arXiv:2106.08948*, 2021.
[3] J. Kupoluyi, "Muzeel: A dynamic event-driven analyzer for javascript dead code elimination in mobile web," 2020.
[4] "A framework for measuring business processes based on GQM," in *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, 2004, pp. 10 pp.–.
[5] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering - An Introduction.* Kluwer Academic Publishers, 2012.
[6] S2-group, "Lacuna-evaluation/experiment on lacuna performance - online appendix .pdf at main · s2-group/lacuna-evaluation," Aug 2022. [Online]. Available: https://github.com/S2-group/Lacuna-evaluation/blob/main/Experiment%20on%20Lacuna%20performance%20-%20online%20appendix%20.pdf
[7] T. D. Cook and D. T. Campbell, *Quasi-experimentation: Design & Analysis Issues for Field Settings.* Wadsworth Publishing Company, 1979.