

MASTER

LUNA

an Interactive Visualization of Javascript Software with Libraries

van Dijk, Roy E.L.

Award date:
2022

[Link to publication](#)

Disclaimer

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain



Department of Mathematics and Computer Science
Software Engineering and Technology Research Group

LUNA: an Interactive Visualization of Javascript Software with Libraries

Master Thesis

R.E.L. van Dijk

Supervisors:
Prof. Dr. Michel R.V. Chaudron
Dr. Eleni Constantinou

Eindhoven, October 2022

Abstract

With software development, how much of the code written by the developers actually ends up in the final application? A large portion of the application's source code is actually made by third party developers, usually through so-called open-source libraries. Software libraries have become commonplace in software development. They are fantastic because they can abstract and simplify code and programming problems. However, it is difficult to maintain an overview of these libraries in the software architecture, especially when they are used in high frequently. This thesis proposes a visualization tool that can show an abstract overview of how libraries interact with software. We present LUNA, or Library Usage in Node.js Analyzer. It is a library-focused software development tool for node.js projects. As the name implies, LUNA is a JavaScript application for the Node.js and NPM ecosystem. This ecosystem is well-known for its abundance of micro-packages and complex web of dependencies. The goal of LUNA is to help developers comprehend how libraries are utilized in their projects. We explore how LUNA can recover the software architecture and library usage by analyzing the abstract syntax tree (AST) of the source code, and create a visualization that can display all this information in an interactive graph. Finally, the utility of LUNA is investigated through user interviews. The results show that the tool can recover the program architecture and library usage, and that the visualization can display this information in a comprehensible manner. The interviews also showed that LUNA can indeed assist developers with better understanding their program architecture and library usage, as is concluded by this thesis.

Preface

I am a master student of Computer Science and Engineering. One of my beloved hobbies is contributing to open source software, of which I've also created many. A large part of the open source ecosystem are software libraries and I personally have used them countless times too. Hence, they are the inspiration behind my thesis. I aim to target fellow software engineers with this thesis. Because the work was carried out entirely by myself, several portions of this article are written in first person. However, I had great aid by my supervisors Michel Chaudron and Eleni Constantinou. I would like to sincerely thank them for their support! Our weekly meetings helped guide me during my master thesis. Finally, I would like to thank Fernando Paulovich for being part of the assessment committee. I sincerely hope you enjoy reading my thesis.

Contents

Contents	vii
List of Figures	ix
List of Tables	xi
1 Introduction	1
1.1 Problem description	1
1.2 Relevancy	1
1.2.1 Software Engineers	1
1.2.2 Library Developers	2
1.2.3 Security Experts	2
1.3 Research questions	2
2 Related Work	5
2.1 Background	5
2.1.1 Software Architecture	5
2.1.2 Software Libraries	5
2.1.3 Visualization	6
2.2 Literature	6
3 Methodology	9
3.1 Approach	9
3.2 JavaScript	9
3.3 Objective	10
3.4 Abstract Syntax Tree	10
3.5 Analysis	12
3.6 Report	13
3.7 Graph	13
3.7.1 Cytoscape	13
3.7.2 Extensions	13
3.7.3 Source Code Component	13
3.7.4 Libraries Component	14
3.7.5 Dependency Graph Component	14
3.7.6 Optimization	14
3.8 Graph Layouts	14
3.8.1 Dage	15
3.8.2 Breadthfirst	15
3.8.3 Cola	16
3.8.4 Cose-Bilkent	17
3.8.5 Elk	17
3.9 Interface	19

3.9.1	Graph Menu	19
3.9.2	Node Menu	20
3.10	Testing	21
4	Results	23
4.1	Research question 1 (RQ1)	24
4.2	Research question 2 (RQ2)	25
4.3	Research question 3 (RQ3)	25
4.4	Main research question (RQ0)	26
5	Discussion	29
5.1	Comparison with Similar Projects	29
5.1.1	CodeGraph	30
5.1.2	HUNTER	31
5.1.3	Eunice	31
5.1.4	NPMGraph	32
5.1.5	JSCity	33
5.1.6	MetropolJS	34
5.1.7	js2flowchart.js	35
5.1.8	JSClassFinder	36
5.1.9	Source Code Explorer	37
5.2	Implications	38
5.2.1	Researchers	38
5.2.2	Practitioners	38
5.3	Threats to validity	39
5.3.1	Construct	39
5.3.2	Internal	39
5.3.3	External	39
5.4	Future work	39
5.4.1	Scanner	40
5.4.2	Visualization	40
5.4.3	User Interface	41
5.4.4	Bugs	42
5.4.5	Beyond LUNA	42
6	Conclusion	43
	Bibliography	45
	Appendix	49
A	MoSCoW	49
A.1	Constraints	49
A.2	Project	50
A.3	GUI	51
A.4	Graph	51
A.5	Node	53
A.6	Information	53
A.7	Scalability	54
B	Abstract Syntax Tree	55
C	Form	77
C.1	Questions	77
C.2	Answers	90

List of Figures

3.1	The very first sketch of the tool	9
3.2	A diagram of LUNA's entire system architecture.	10
3.3	Imports of the scanner.js shown in LUNA's visualization.	12
3.4	The three components of LUNA's visualization (collapsed).	13
3.5	A layout algorithm for the graph in LUNA's generated report: Breadthfirst	16
3.6	A layout algorithm for the graph in LUNA's generated report: Cola	16
3.7	A layout algorithm for the graph in LUNA's generated report: Cose-Bilkent	17
3.8	The default layout algorithm for the graph in LUNA's generated report: Elk (Layered)	18
3.9	A layout algorithm for the graph in LUNA's generated report: Elk (Mr. Tree) . .	18
3.10	Graph menu in LUNA's generated report	19
3.11	File node menu in LUNA's generated report	20
3.12	Library node menu in LUNA's generated report	21
4.1	The description of each task, that the interviewees had to perform.	23
4.2	Some of the results of each task.	24
5.1	Overview of CodeGraph.	30
5.2	The GUI of Hunter	31
5.3	Eunice's HTML report.	32
5.4	The GUI of NPMGraph.	33
5.5	Screenshot of JSCity.	34
5.6	MetropolJS in action	35
5.7	Demonstration of js2flowchart.js	36
5.8	A Class Diagram Generated by JSClassFinder (for JSClassFinder's algorithm.js) .	37
5.9	The GUI of the Source Code Explorer	38

List of Tables

3.1	Comparison of different graph layouts used by LUNA.	15
4.1	The combined SUS test results.	26
5.1	Comparison between LUNA and similar tools.	29

Listings

3.1	Snippet of the LUNA's scanner.js showing how files and libraries are imported via RequireJS.	11
B.1	AST of the snippet of the LUNA's scanner.js	55

Chapter 1

Introduction

The topic of this master's thesis is introduced in this chapter. We begin by describing the problem at hand as well as the reasons why it is relevant. After that, the research questions are stated and discussed.

1.1 Problem description

The term *software package* has several meanings [1]. It is used to refer to a library or framework in this thesis, which is a collection of code written to be reused by other software applications. So, a software library's primary function is to offer a solution to a particular problem or set of problem, which can be accomplished by offering a set of methods, data structures, or classes. It can be thought of as a black box. Libraries can substantially help with software development. Typically, they can be imported and used anywhere in the source code of the application. However, where does it make the most sense to do in a software architecture? And what are software doing in practice? How can you manage and keep track of all the libraries that are used? We don't really have a straightforward answer for these problems, but this would be interesting to know in order to make good software design decisions relating to libraries. The objective of this thesis is therefore to comprehend and illustrate how software packages are employed in the software architecture. This is accomplished through an investigation into how software libraries are used in software. For this, a tool named LUNA is created, which is covered extensively in chapter 3. An interactive demo of it can be found online at <https://royvandijk06.github.io/luna>.

We will now first discuss the problem's relevance and identify the stakeholders. Afterwards we go over the research questions and explain why they are important to the research problem.

1.2 Relevancy

As mentioned earlier, the goal of this thesis is to make us more informed about how libraries are integrated in the software codebase. This is relevant to both researchers and practitioners. Researchers can utilize the tool developed for this thesis for other research or to extend this research (section 5.4). Practitioners, like software developers and maintainers, can use the results of this research to make more insightful design decisions when incorporating libraries. The research problem from section 1.1 concerns several stakeholders. Let us consider what use cases these stakeholders may have for software packages and what they may gain from this research.

1.2.1 Software Engineers

1. They can evaluate the impact a library has on their project (in terms of usage frequency)

2. They can reduce the complexity of their project by using libraries that provide simplification or abstraction
3. They can showcase information about used libraries to other interested parties
4. They can find out what needs to be changed in the project code when replacing libraries

1.2.2 Library Developers

1. They can see what parts of their library is being used in a project
2. They can see what parts of the project is using their library
3. They can comprehend the tree of dependencies of their library

1.2.3 Security Experts

1. They can find out where in a project a vulnerable or compromised library is being used
2. They to what extent a vulnerable or compromised library is being used in a project

1.3 Research questions

In this section, we will go over the research questions related to the problem described in section 1.1. There is one question central to the problem described in section 1.1, which is the following:

RQ0 *How to facilitate the comprehension of library usage in a software architecture?*

Hence, I am proposing this as my main research question. However, since this is a rather broad research question, a handful of sub-questions are proposed to help answer the main research question:

RQ1 *How to recover the software architecture?*

It is essential to understand how to recover the software architecture because we need to know what the architecture looks like in order to answer the primary research question (RQ0). This is also relevant to the remaining research questions. Before we can recover software architecture, we must first analyze what defines it. This is done in section 2.1.1 and in the threads of validity from section 5.3, where we discuss the meaning of software architecture and whether our interpretation is valid.

RQ2 *How to detect libraries and their usage?*

Before we can answer the main research question, we need to know how to detect libraries and their usage. The architecture of the software plays a role in this, since we need to know where to look for libraries. Hence, this question is related to the previous sub-question (RQ1).

RQ3 *How to visualize this information in a useful way?*

Finally, we must understand how to visualize the information that was gathered in order to facilitate the comprehension of the utilization of libraries in the software architecture. This information follows from both RQ1 and RQ2. Note that this research question basically asks two things: how to visualize software architecture with libraries, and how to accomplish this with usefulness in mind. A visualization is meaningless if it can not be useful to the user.

The answers of these sub-questions together should be able to answer the main research question of this thesis. The research questions are formulated in such a way that they can be answered by the tool developed in this thesis, and the qualitative research that follows.

The remainder of this paper is organized as follows. In chapter 2, related work is presented. Next, in chapter 3, the methodology is described of the study presented in this paper. Then, chapter 4 includes the answers to the proposed research questions. Comparison with related studies, the implications in the field and threats to validity are discussed in chapter 5. Finally, I conclude the paper in chapter 6.

Chapter 2

Related Work

Before we delve into the methodology used for this thesis research, let us explore related work in the domain of software libraries and architecture, as well as visualization.

2.1 Background

In this section, we familiarize ourselves with the concepts used throughout this paper. Background information is given for the topics of software architecture, software libraries, and visualization.

2.1.1 Software Architecture

The foundational structures of a software system, as well as the discipline of building such structures and systems, are referred to as software architecture. Each structure is made up of software components, relationships between them, and attributes of both elements and relationships [2]. These components can be modules, classes, or functions, and the relationships can be dependencies, associations, or inheritance. The attributes can be properties of the components, such as their name, or properties of the relationships, such as their type. The software architecture is a model of the system, and it is used to describe the system's structure and behavior. The architecture is also used to guide the development of the system, and to communicate the system's structure and behavior to stakeholders [3].

A good software architecture is essential for a variety of reasons: it makes the code easier to write, understand, and modify; it can make the code more maintainable and bug-free; and perhaps most importantly, good software architecture makes it possible to create scalable applications. In other words, with a well-designed architecture in place, an application can easily handle increased loads without breaking down or becoming unstable. Conversely, a poorly designed architecture will likely lead to an unmaintainable mess of code that is very difficult (if not impossible) to scale.

If we want to analyze the architecture of software, we need to recover it. The basic idea is to analyze some structural information of the software, e.g., its code or dependencies, and try to infer from this analysis what the high-level organization of the software looks like. There exist many techniques for software architecture recovery. In section 2.2, research on comparing software architecture recovery techniques will be discussed [4] [5].

2.1.2 Software Libraries

A software library can be defined as a collection of subroutines and functions that are used to perform common tasks. It is a set of code that can be reused in different programs. We call the provided functionality by the library Application Programming Interface (API), which is how the library communicates with the rest of the application. When you write a program, you can use these libraries without having to write your own code from scratch. This saves you time and effort because you don't have to reinvent the wheel every time you need a certain functionality. Software

libraries are important because they provide building blocks for programmers. They allow us to break down complex problems into smaller pieces and then put them back together again into working solutions. Without libraries, we would have to start from scratch every time we wanted to create something new.

2.1.3 Visualization

Visualization, or more specifically data visualization, is the presentation of data in a pictorial or graphical format. It is a way of communicating information by encoding numbers, symbols, and images into visual objects. The goal of data visualization is to communicate information clearly and efficiently through statistical graphics, plots, information graphics, and other visual formats. Data visualization is a powerful tool for communicating information. It allows us to see patterns and trends in data that would otherwise be difficult to detect. It also allows us to make predictions about the future based on past data. Data visualization is an important part of data science because it allows us to understand data in a way that is not possible with just numbers and text. There exist a wide variety of data visualization techniques, such as scatter plots, bar charts, and pie charts. In this thesis, we will focus on graph visualization techniques [6]. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. It is a way of representing information in a way that is easy to understand and interpret.

2.2 Literature

This section is devoted to the literature that is relevant to this thesis, covering the following topics: software architecture recovery, software libraries, and visualization. This literature is used to provide background information for the methodology used in this thesis. Each work will be briefly introduced and described, as well as the similarities and differences between them, and how they relate to the thesis's research problem.

For starters, in Kula et al. [7], the impact of micro-package in an open source software (OSS) ecosystem is investigated, specifically the JavaScript NPM ecosystem, which is also a relevant subject that is discussed in chapter 3. As demonstrated later, the tool developed for this thesis can clearly demonstrate this issue for the NPM ecosystem. In this work, micro-packages are defined as minimized libraries, and they can have long dependency chains which may become problematic for critical systems.

Again, the software package manager NPM is investigated in the works of Zerouali et al. [8]. Specifically, they research the meaning of popularity metrics that are used for software libraries and compare the existing ones.

The paper of Žitný et al. [9] is an example of a master thesis that analyzes NPM and JavaScript at scale. Among other things, they discuss the reasons for the high percentage of clones and present few ideas on what analyzes can be done to in the future with collected data.

Len Bass et al. [3] wrote an interesting book about software architecture in practice. The book describes software architecture in detail, including why it is important and how to design, instantiate, analyze, evolve, and manage it in a disciplined and effective manner.

One way to analyze software architecture, is to detect common design patterns. This is exactly what was done by Bautista et al. [10] as a step for documentation generation for JavaScript projects. As their methodology, they used fingerprinting on a constructed abstract syntax tree (AST) of the codebase. The research of Taraghi et al. [11] also focuses on documentation generation, specifically for Node.js web applications. They combine automated class diagram generation with some manual adjustments, alongside source code analysis and informal interviews, as a software reverse engineering method to output two documents: Software Requirements Specification and Software Design Document. The tool developed for this thesis also analyzes the AST of the codebase, but it does not focus on documentation generation. It also provides minimal class abstraction.

Another paper related to architecture recovery is the paper of Nurwidyantoro et al. [12], where they present an automated machine learning-based approach for classifying the role-stereotype of classes in Java. Additionally, they compare their approach to another existing rule-based classification approach.

There exists multiple software architecture recovery techniques, Agarwal et al. [4] proposes a research to compare all of them. They use software testing tools to compare recovery algorithms among multiple projects. Likewise, the paper from Garcia et al. [5] compares the performance of several software architecture recovery techniques by using a set of 8 open source projects and their architecture ground truths.

Software libraries update over time, which may result in breaking changes to their available API. In Xavier et al. [13] they performed a large-scale empirical study investigating API breaking changes and their impact on client applications. Their study finds that (i) 14.78% of the API changes break compatibility with previous versions, (ii) the frequency of breaking changes increases over time, (iii) 2.54% of their clients are impacted, and (iv) systems with higher frequency of breaking changes are larger, more popular, and more active.

Similarly to the previous work, the work of Ochoa et al. [14] also researched the impact of breaking changes in API, but unlike the work of Xavier et al. [13] that was previously mentioned, they focus specifically on libraries and take their evolution and semantic versioning into account. They developed a tool called Maracas with the goal to detect breaking changes between 2 versions and what parts of the client code is affected by them.

In the paper from Mili et al. (1995) [15] they discuss the implications of reuse of software in the production. Later they performed a survey on the subject and published the results in Mili et al. (1998) [15]. This gives us a good insight on the origin of software libraries and how they came to be.

Many methods for representing software components for reuse exist and in the work of Frakes et al. (1990) [16] these methods are surveyed and categorized. Additionally, they discuss systems in which they have been used and propose a framework of software reuse representation that relates these methods. Later they performed an empirical study on this in Frakes et al. (1994) [17].

In the paper of Feldthaus et al. [18] they discuss the difficulties of constructing call graphs for JavaScript. Call graphs capture the connectivity between software functions. They tackle this problem by presenting a scalable field-based flow analysis for constructing call graphs which proved to work well in practice.

The work of Nielsen et al. [19] also present a novel solution for call graph construction in JavaScript, specifically for the node.js ecosystem. They approach the problem from a security point of view. Additionally, they take into account the modular structure of Node.js applications. They claim to be more accurate and efficient compared to competitor software.

Tarawaneh et al. [6] gives us a general introduction to graph visualization techniques. It gives a general overview about several layout algorithms and interaction techniques.

Chapter 3

Methodology

In this chapter, we explore the methods used to tackle our research questions from section 1.3. We first discuss the general approach and then what constraints and objectives are set. We then go over the tool's design and the design decisions that were made.

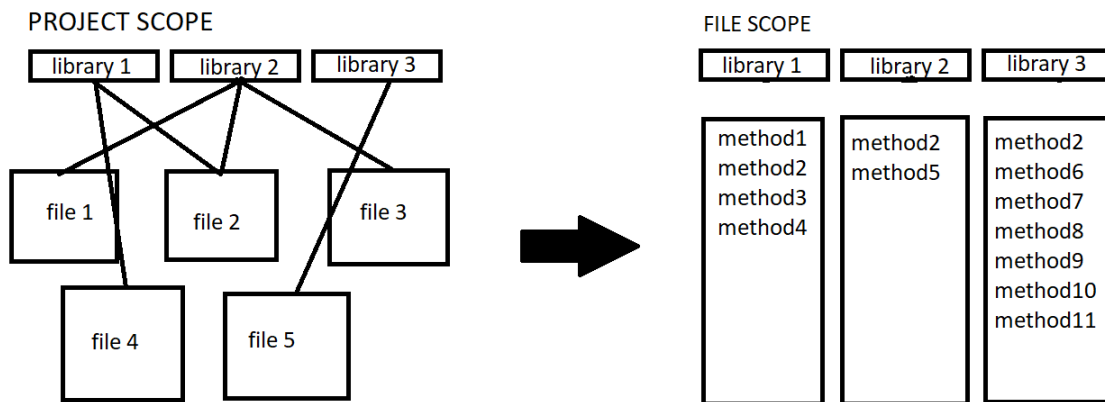


Figure 3.1: The very first sketch of the tool. A view that connects libraries with project files (left) and a view that lists library API utilization per file (right).

3.1 Approach

For our research problem, we want to find ways to indicate the usage of libraries in any software project in a way to make it better comprehensible. Analysis and visualization are the most effective ways to accomplish this. To actually achieve these two components, a tool can be built. In section 3.3, the objectives of this tool are stated. I created it using iterative software development, where it is iteratively improved upon. The tool's name is **L**ibrary **U**sage in **N**ode.js **A**nalyzer, abbreviated as **LUNA**.

3.2 JavaScript

JavaScript is a programming language and is, alongside HTML and CSS, one of the core technologies of the World Wide Web. Nowadays, nearly all websites use JavaScript to control the webpage behavior. The code is executed by a dedicated JavaScript engine, which all major web browsers have. JavaScript conforms to the ECMAScript standard. It is high-level and has dynamic typing, prototype-based object-orientation, and first-class functions. It is multi-paradigm, supporting

event-driven, functional, and imperative programming styles.

I will focus on analyzing the JavaScript programming language, particularly the node.js runtime environment. Because it is well-known for having a large number of third-party libraries and because it is well-known to me (years of practical experience). The libraries in this ecosystem are better known as NPM (Node Package Manager) modules, but we will remain calling them libraries, or dependencies, i.e., the libraries that a project or library depends on.

3.3 Objective

As explained in section 3.1, my approach includes building a project scanner for analysis, the first main component of LUNA. This scanner will detect all the libraries used in a project. For each library it will also indicate the usage, meaning:

1. How much of the library functionality is being used per file, in terms of the available API (methods & data);
2. Where it is used, in what files and in what software components (functions & classes);
3. Whether libraries are external (from third-party developers) or internally developed

Software architecture detection techniques are used to locate places in the architecture where libraries are utilized. More details on that can be found in section 3.4. Additionally, visualization will be the second main component of LUNA. In appendix A a full list of software requirements for LUNA can be found.

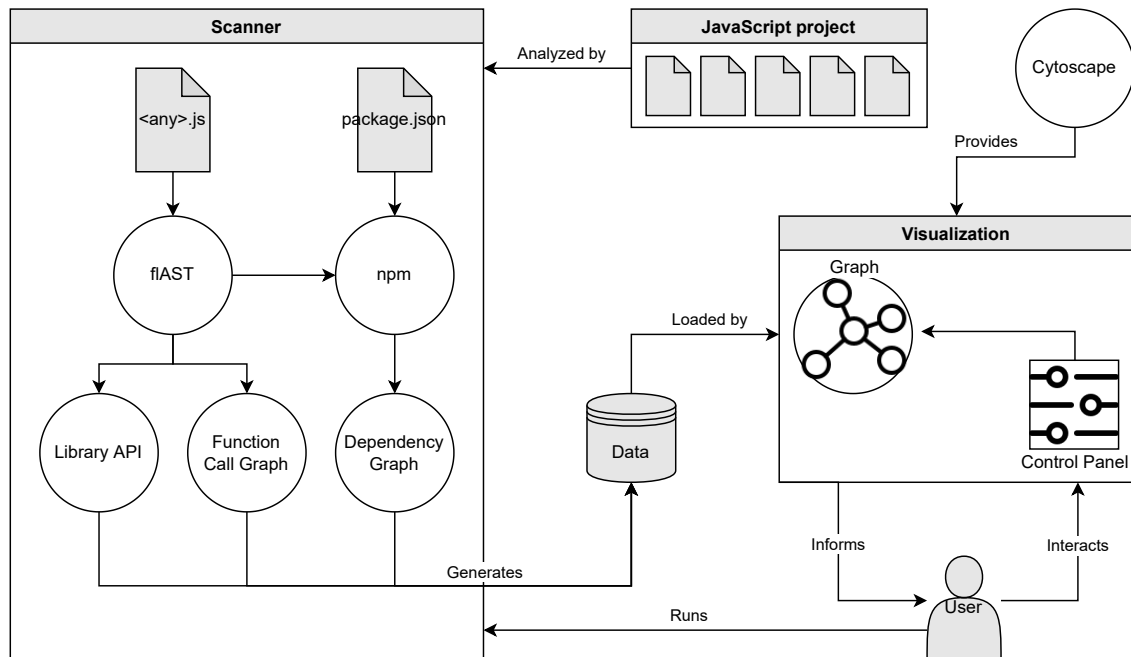


Figure 3.2: A diagram of LUNA's entire system architecture.

3.4 Abstract Syntax Tree

To properly analyze source code, an Abstract Syntax Tree (AST) is crucial, which is a tree representation of the source code. Each node of the tree denotes a construct occurring in the source code. The AST is a static representation of the code, meaning that it does not change

during runtime. It is a way to represent the structure of a program in a way that is easy to analyze. For producing the AST, I initially choose Acorn [20], as this is a well-established parser for JavaScript and is still being actively maintained. One key feature I required for proper analysis, was the ability to find references to a variable in the code. I found a third party library (scope-analyzer [21]) to do this. Unfortunately, during development I found that scope-analyzer [21] was inaccurate and this would not be fixed anytime soon¹. Roughly, at the same time I took note of a new promising project that offered the same kind of features, but using a different underlying AST parser. Although, it was fairly late in the project, it was decided to rewrite some parts of the codebase to implement this new library, called *flast* [22]. This is essentially a wrapper for *eslint-scope* [23] that outputs a flattened AST object that includes scope and reference information for each node. It uses a more modern parser that was built on top of Acorn [20], called *espre* [24].

```
1 // Internal dependencies
2 const { constructString, constructTemplateLiteral, findReferences } =
  require("./common");
3 const { extractCalls } = require("./call-graph");
4 const { extractLibs } = require("./library-api");
5 const { getNodeModules } = require("./dependency-graph");
6
7 // External dependencies / libraries
8 const { basename, dirname, extname, relative, resolve } = require("
  path");
9 const { generateFlatAST } = require("flast");
10 const { promisify } = require("util");
11 const { readFile, stat } = require("fs/promises");
12 const glob = require("glob");
13 const randomColor = require("randomcolor");
```

Listing 3.1: Snippet of the LUNA's scanner.js showing how files and libraries are imported via RequireJS.

¹<https://github.com/goto-bus-stop/scope-analyzer/issues/32>

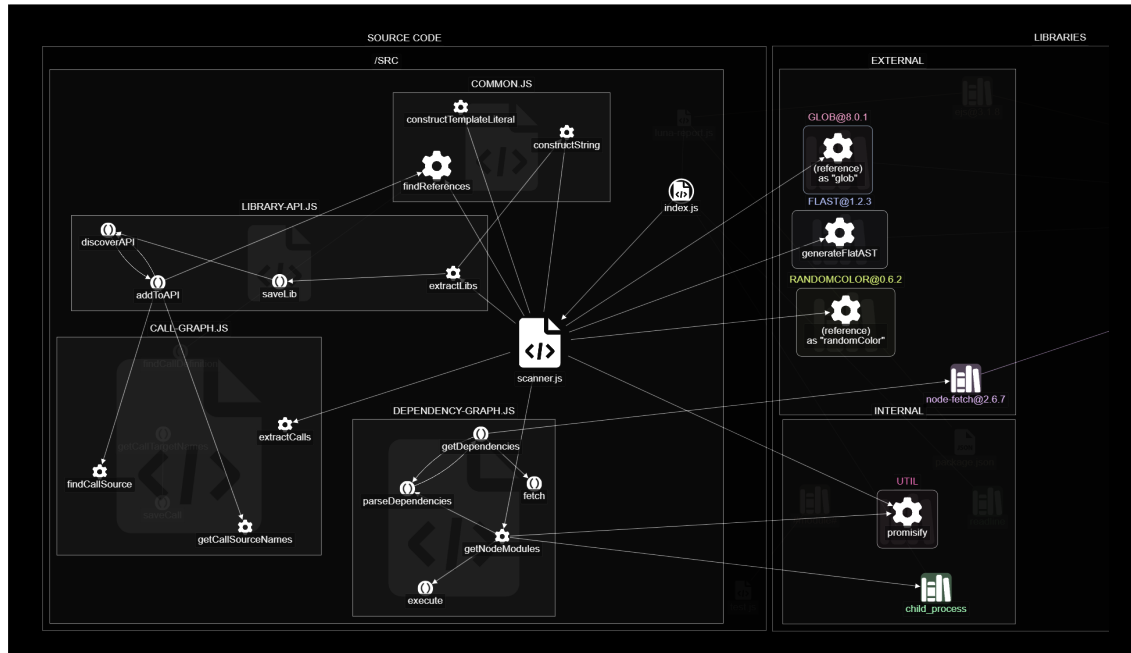


Figure 3.3: Imports of the scanner.js shown in LUNA's visualization.

3.5 Analysis

For the analysis, function call chains and software library API usage have to be tracked, as well as the tree of dependencies. At first, discovery on GitHub and NPM was performed to find existing solutions. Unfortunately, many of them were lacking, outdated, incompatible or simply not working. In the previous section (section 3.4), the solutions are listed that ended up being used, yet most of the analysis tasks still had to be designed manually. Let us walk through this implementation.

When scanning a project, it first constructs a dependency tree object by either invoking NPM's [25] native method of listing all installed dependencies, or as fallback (for when these dependencies are not locally installed), it recreates this dependency tree object by recursively requesting a web API for each parent dependency that lists its dependencies. After that is accomplished, it reads all JavaScript files of the project. However, it will ignore the files of the installed dependencies located in the `node_modules` folder. It then will construct an AST for each read file, as described in section 3.4. This AST is used to extract information about the code structure and used libraries. It generates a function calling tree of the file by detecting any function calls and tracing its source and destination/target, i.e., the function definition. Additionally, it tries to detect all the imported libraries and their provided features/functionality, i.e., the API of the library. They are computed based on the `require` and `import` declaration nodes of the AST. This will detect both the declarations made at the beginning of the JavaScript files and the declarations nested within the source code. Finally, all the collected information is transformed into a data object that can be imported by the graph located in the report. This process is depicted in diagram 3.2. As an example, listing 3.1 is a snippet of code, listing B.1 shows its AST, and lastly figure 3.3 displays the visualization of it from LUNA.

3.6 Report

After LUNA has completed scanning the project and generating the data needed for the graph, it generates a report. The report format is HTML and may be viewed by any Chromium-based web browser, such as Google Chrome. This report has also SVG-based images, JavaScript scripts and CSS stylesheets embedded. The report HTML is rendered using a template engine called EJS [26]. This solution was chosen to embed all data and code in the report. This made the generated report portable, so it can be shared between machines (or even be hosted on a static website).

3.7 Graph

A graph was chosen as the main way to visualize library usage in a project. Nodes may represent different elements of the source code, such as directories, files, libraries, classes and function calls. Edges are used to represent utilization or function calling, i.e., an arrow from A to B, when A uses/calls B. Let us take a closer look at how this graph is constructed and what components it is made of.

3.7.1 Cytoscape

LUNA makes use of a library called cytoscape.js [27], which is an open-source library for fully-featured graphs, written in pure JavaScript. It is a powerful library that allows for extensions and a lot of customization. Furthermore, it has good documentation, which makes it easy to work with. This library is used to generate the graph in the report. The graph is created using data that is generated by the analysis part of LUNA, see section 3.5. The graph displays 3 components as sub-graphs: source code, libraries and dependency graph. Each component visualizes something different. The components are also interconnected.

3.7.2 Extensions

As mentioned, cytoscape.js [27] has support for extensions. LUNA uses a few of these extensions. One of these extensions is used to add extra functionality to the graph, all others are used for the graph layout and are covered in section 3.8. To support collapsing and expanding of nodes, LUNA uses the cytoscape-expand-collapse [28] extension for cytoscape.js. This is useful for large graphs, as it allows for a better overview by collapsing irrelevant parts. Hence, it allows for a better focus on a specific area of the graph.

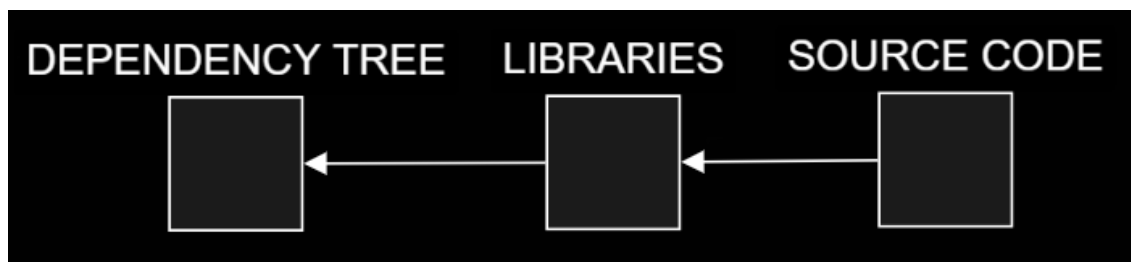


Figure 3.4: The three components of LUNA's visualization (collapsed).

3.7.3 Source Code Component

This component visualizes the structure of the software project. Not only does it display the file and directory structure, something that is often carefully designed by the project architect, it also allows files to be expanded into a function call-graphs, as discussed in section 3.5. However, the nodes representing files are collapsed by default. Furthermore, the size of file nodes are determined

by the amount of lines of code in the file. This helps users to quickly identify the most important files in the project, or at the very least the files with the most amount of code.

3.7.4 Libraries Component

This component list all the libraries connected to the software project. Library nodes can be expanded to expose their used API, i.e., functions or objects/classes containing methods. They are divided into 3 groups. One group consist of all the internal libraries, such as the native modules provided by node.js or libraries that are internally developed. Another group lists all the external libraries, which are the third party libraries. The last group includes libraries that are not directly used by the codebase itself but are likely only used for the development of the project. Each library has their own randomized color, so it is easier to track a specific library. Anything connected to this library shares the same color.

3.7.5 Dependency Graph Component

This component includes the dependency chain of all the libraries from the libraries component (section 3.7.4). Because these dependencies are nodes that are connected to a library node, they share the same color. The entire dependency chain shares the same color, unless it is connected to another dependency chain, because one dependency may be included in more than one dependency chain. This component is often a big complex network of dependency nodes, so not to overwhelm the user, this graph component is collapsed by default.

3.7.6 Optimization

With the additions of file call-graphs and library API nodes during LUNA's development, the amount of nodes and edges to load into the graph became exponentially greater. In some bigger projects, it even became impossible to load. So, some kind of optimization was required. Fortunately, a large portion of nodes are hidden behind collapsed groups of nodes or sub-graphs, as discussed in the previous sections about the graph components. Therefore, during the initial load of LUNA's graph, these nodes and edges can be omitted. The challenge was to omit the correct nodes and edges, and add the correct nodes and edges back upon an expansion event of their parent node. Additionally, nodes that connect to other nodes outside their parent could not be omitted without breaking the collapsing functionality. Fortunately, this on-demand loading technique had been successfully achieved and drastically improved the loading times compared to the non-optimized version.

3.8 Graph Layouts

The position of the nodes that are displayed in a graph are computed by a layout algorithm. In this section we will discuss the layout algorithms that were considered or implemented in LUNA. Table 3.1 shows a summary of the different layouts and their strengths and weaknesses, based on personal experience with the graph layouts.

Layout	Figure	Edge crossing	Node overlap	Hierarchy & ranking	Grouping & clustering
Breadthfirst	3.5	0	–	+	0
Cola	3.6	–	–	–	+
Cose-bilkent	3.7	–	0	0	0
Elk: Layered	3.8	–	+	–	–
Elk: Mr. Tree	3.9	0	–	+	0

Table 3.1: Comparison of different graph layouts used by LUNA. A layout algorithm’s strength is indicated by +, weakness is indicated by – and 0 means neutral.

3.8.1 Dagre

Dagre is a discrete layout that places nodes in a hierarchical order and minimizes the number of crossing links. The general implementation is inspired by ”A Technique for Drawing Directed Graphs” [29] and the method for minimizing the number of crossing links is based on ”2-Layer Straightline Crossing Minimization” [30].

After trying a handful of layouts, this one seemed to be best performing, as it produces a clear and usable layout. Therefore, this layout was initially chosen for the graph. Later in development, some critical bugs were noticed in this layout algorithm. This led to the discovery that the underlying library it relied upon was deprecated². Hence, it was decided to drop the support for this layout algorithm.

Instead, I choose to support a collection of layout algorithms. This collection was curated and tuned to be the best performing of all supported layouts by the graph. However, none of them were perfect for all use cases, so an option for the user to cycle between the layouts in this collection was added to LUNA. That way the user has the ability to use the best layout for their use case or preference.

3.8.2 Breadthfirst

The breadthfirst layout is natively supported by cytoscape.js. It puts nodes in a hierarchy based on a breadthfirst traversal of the graph. This layout algorithm is best suited for tree/forest graphs in its default top-down mode, and for DAGs in its circle mode.

²<https://github.com/dagrejs/dagre#important>

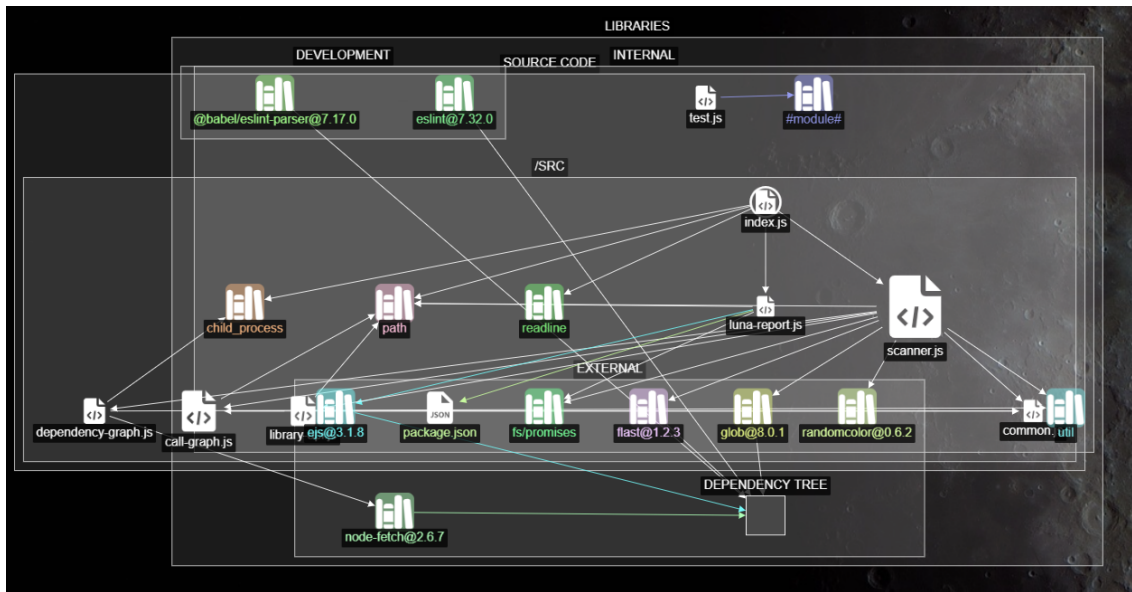


Figure 3.5: A layout algorithm for the graph in LUNA's generated report: Breadthfirst

3.8.3 Cola

This is a layout for cytoscape.js, which uses a force-directed physics simulation with several sophisticated constraints.

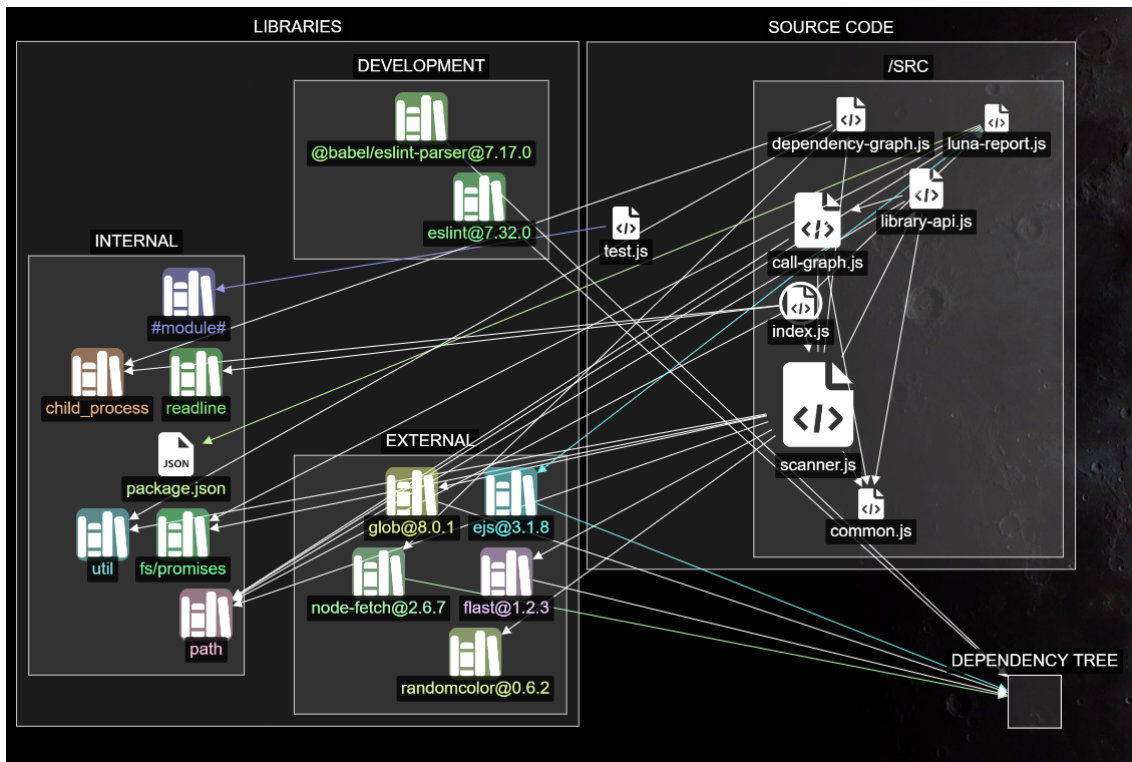


Figure 3.6: A layout algorithm for the graph in LUNA's generated report: Cola

3.8.4 Cose-Bilkent

Cose-Bilkent is a spring embedder layout for cytoscape.js with support for compound graphs (nested structures) and varying (non-uniform) node dimensions.

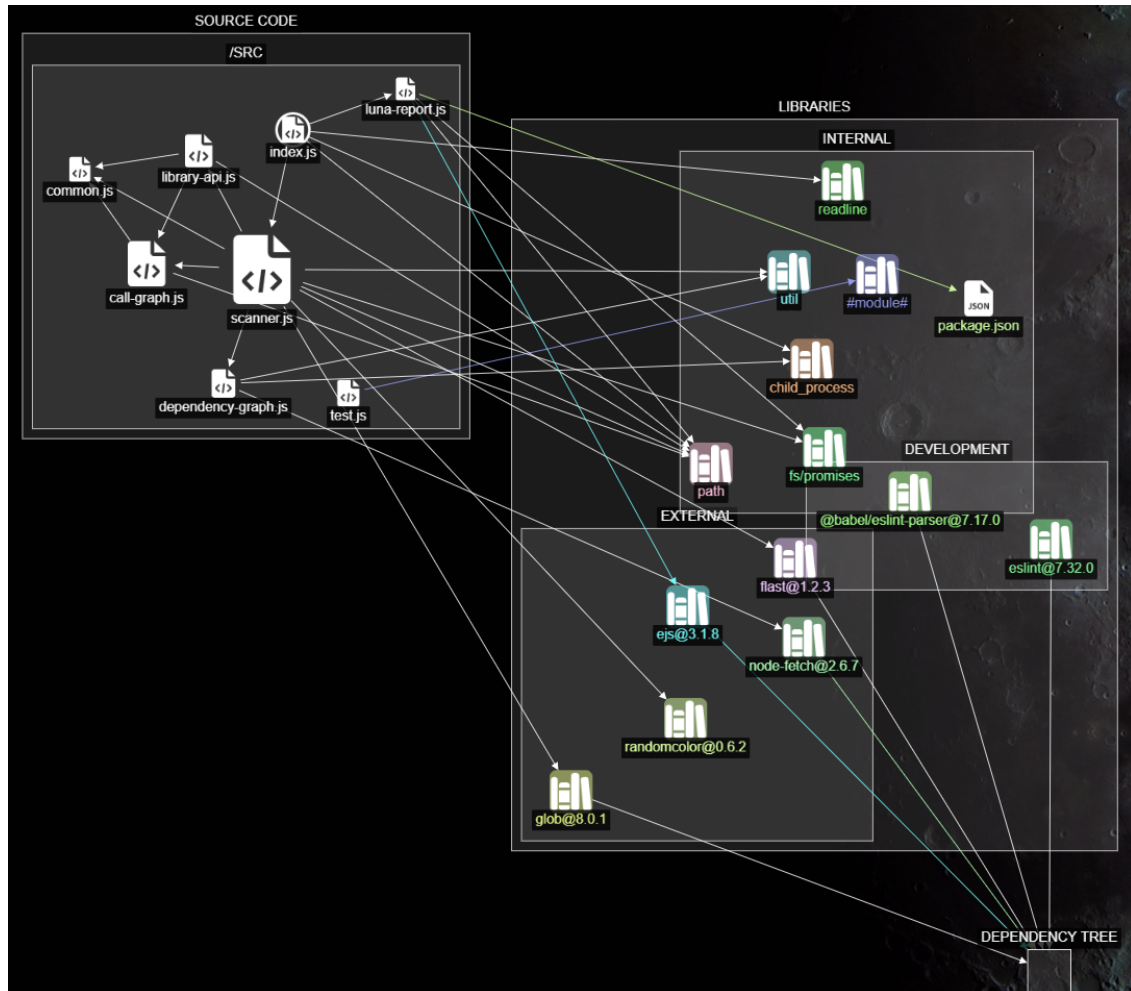


Figure 3.7: A layout algorithm for the graph in LUNA's generated report: Cose-Bilkent

3.8.5 Elk

ELK is a set of layout algorithms implemented by the Eclipse Foundation in Java. Cytoscape.js has a layout adapter for this.

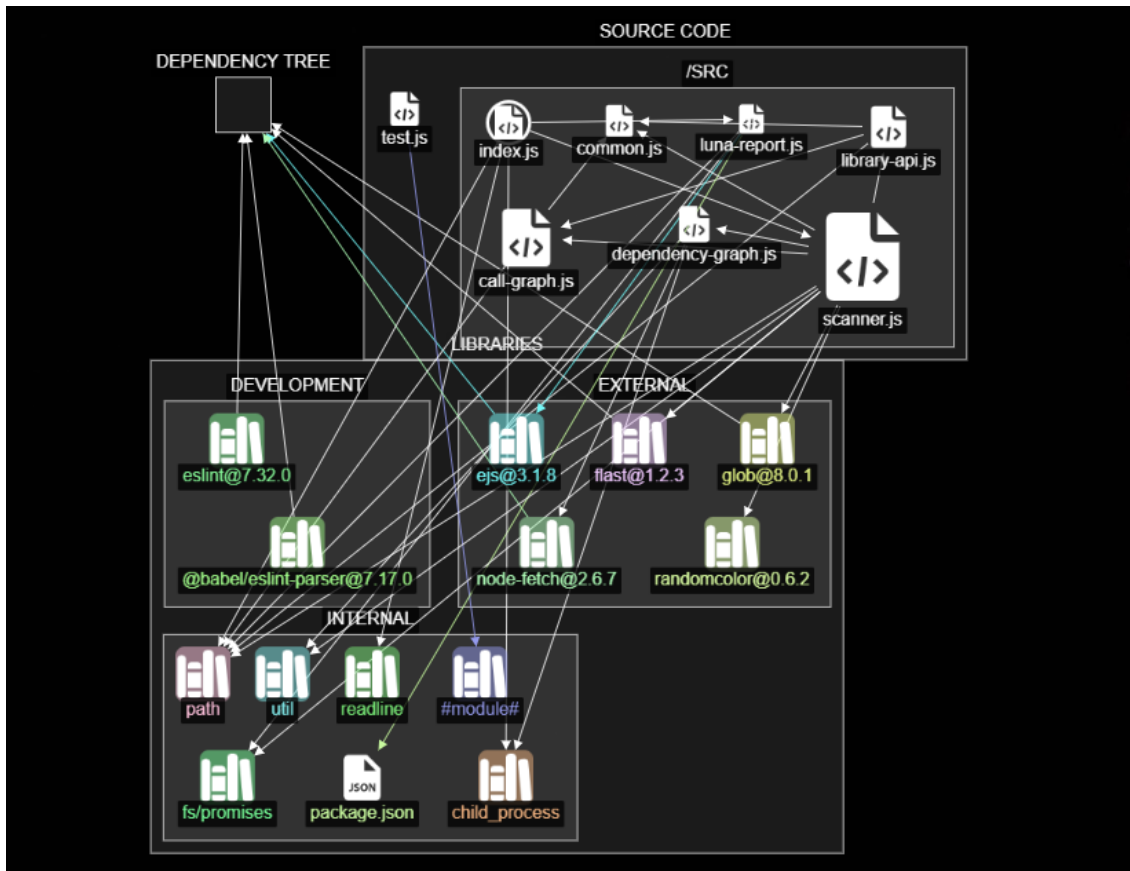


Figure 3.8: The default layout algorithm for the graph in LUNA’s generated report: Elk (Layered)

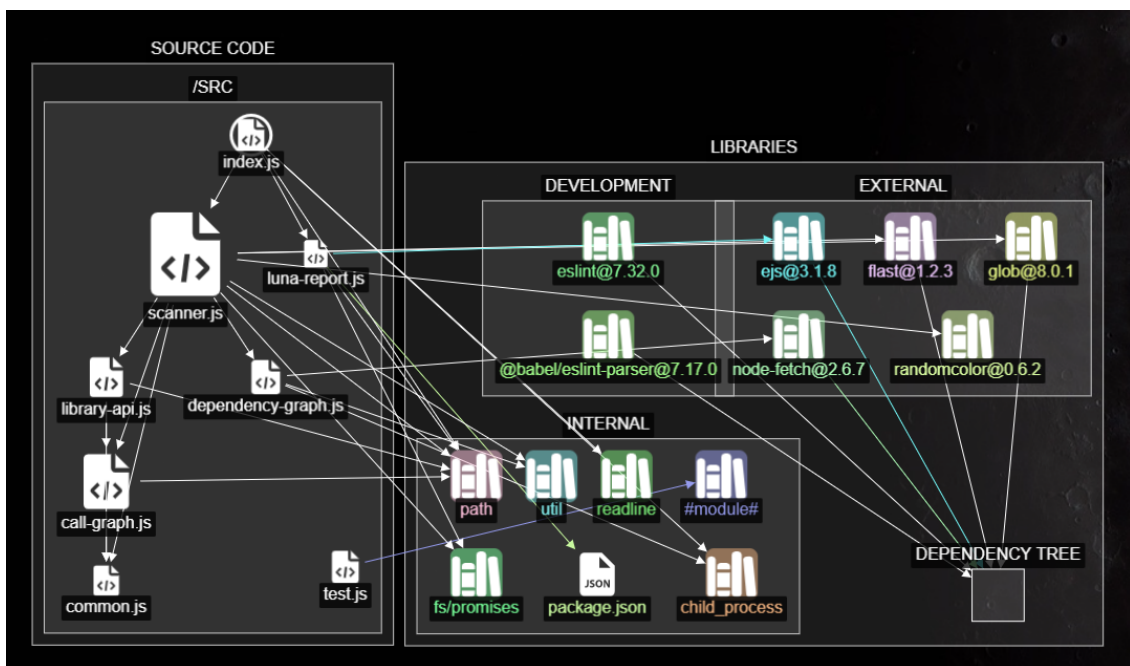


Figure 3.9: A layout algorithm for the graph in LUNA’s generated report: Elk (Mr. Tree)

3.9 Interface

LUNA consists of two parts: the scanner and the visualization. The scanner offers a command-line interface (CLI), which means you can only interact with it via a terminal or console. It does only one thing, which is scan a project and generate a report of it, using the command "npx luna-scanner". The interface of the visualization from the report is significantly more feature-rich. Aside from the graph that was discussed in section 3.7, the main way to interact with the visualization is via the two menus available in the report. Let us take a look at them.

3.9.1 Graph Menu

This menu includes options to interact with the entire graph. Figure 3.10 shows the graph menu and explains the options available in it.



Figure 3.10: Graph menu in LUNA's generated report. Menu elements: A) Toggle visibility of all the JSON data files that are imported; B) Change the distance between nodes via this spacing factor; C) Change the positions of the nodes in the graph dictated by the selected layout algorithm; D) The section to control all the file nodes that exist in the graph; E) Toggle highlighting of a node in the graph; F) A folder that is collapsible (via double click) in the file control section; G) Toggle visibility of a node in the graph; H) A file in the file control section; I) The section to control all the library and dependency nodes that exist in the graph; J) External libraries group; K) Internal libraries group; L) Development libraries group; M) Dependency graph group; N) Take a screenshot of the graph that is in the viewport and save it to disk as 'luna.png'; O) Fit the whole graph into the viewport; P) Search for a node in the graph by label (case-insensitive); Q) Reset the position of the viewport to the center.

3.9.2 Node Menu

This menu show information about a selected node and has options to manipulate it. Figure 3.11 and 3.12 shows the node menu and explains the options available in it.

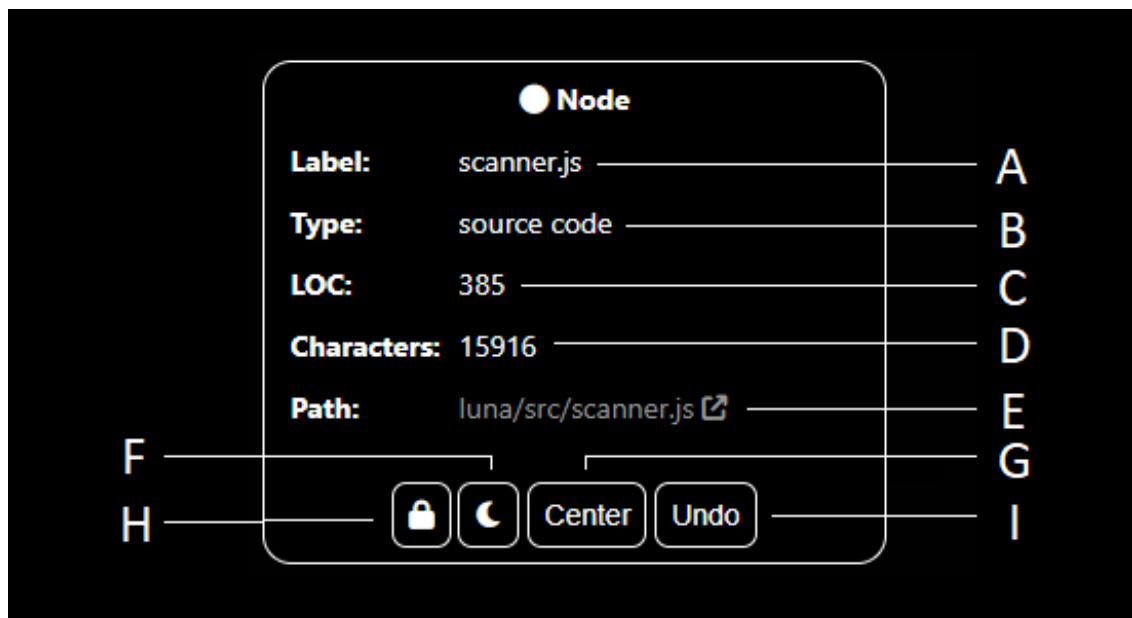


Figure 3.11: File node menu in LUNA's generated report. Menu elements: A) The label of the node; B) The category that this node represents (API, class, file, folder, function call, JSON data or library/dependency); C) Total lines of code in this file; D) Total characters of code in this file; E) Absolute path to this file (click to view raw source code); F) Not used; G) Focus on the node; H) Toggle mouse hover functionality; I) Toggle highlighting of the node.

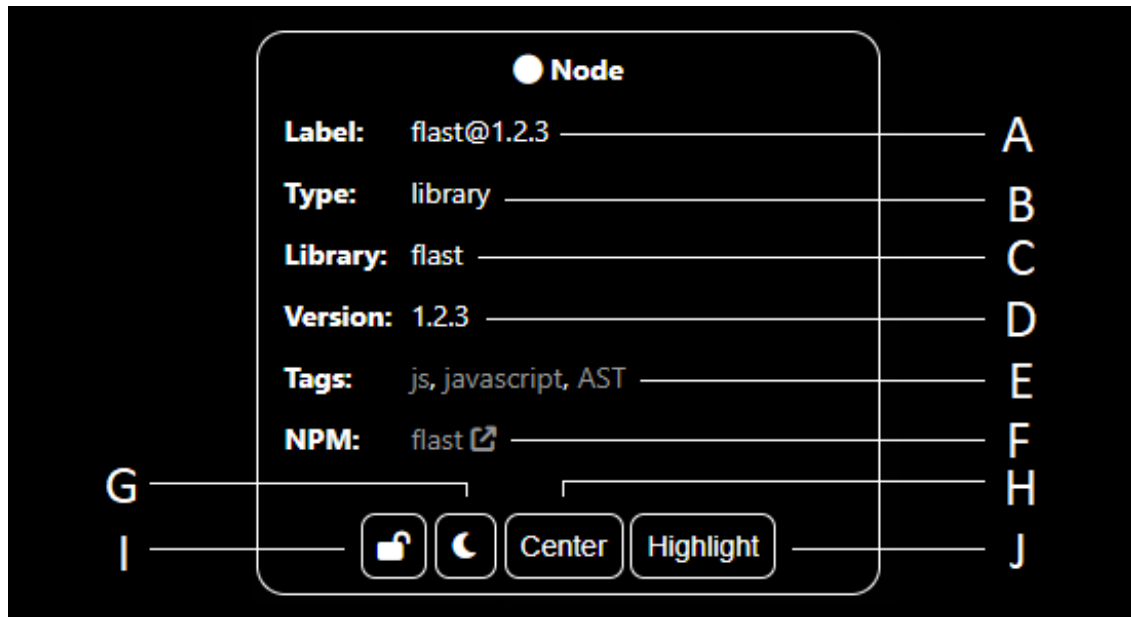


Figure 3.12: Library node menu in LUNA’s generated report. Menu elements: A) The label of the node; B) The category that this node represents (API, class, file, folder, function call, JSON data or library/dependency); C) The name of the library; D) The version of the library; E) Keywords related to the library (click to highlight all nodes with the same keyword); F) Link to the library listing on npmjs.com; G) Button that shows the user a bash command to generate a LUNA report from this library; H) Focus on the node; I) Toggle mouse hover functionality; J) Toggle highlighting of the node.


3.10 Testing

During development of LUNA, tests were performed to ensure proper functionality. A debug mode was added to provide additional logs, as well as enable limited hot-reload functionality for the generated LUNA report, which allowed changes to the client code and styling to be applied after reloading the report without regenerating it. Normally, a LUNA report has all its code and data embedded into a single static file, as explained in section 3.6. Instead, in debug mode, certain scripts and styles are not embedded, but rather linked to the development files of LUNA. Most tests were performed on the LUNA project itself, but to ensure good performance for LUNA across all JavaScript projects, several additional projects were included in the test suite for LUNA. These projects were deliberately chosen to cover a wide variety of the JavaScript development space. For example, one of the projects included in the test suite was the popular jQuery library [31], which has a lot of internal libraries, but no external libraries as dependencies. Another project that was frequently used as a stress-test for LUNA was the source code for NPM’s CLI program [25], which included over five thousand JavaScript files and 2500 folders at the time of testing. LUNA is currently able to handle these big projects, although with a fair bit of struggle and long loading times. Aside from internal testing, ten interviews were performed with friends and colleagues, which let them test LUNA and provide feedback. The results of these interviews are covered in chapter 4.

Chapter 4

Results

In this chapter, the study findings are explained and analyzed in relation to the research questions. We will first go through the supportive research questions (RQ1-RQ3), before we answer the main research question (RQ0).



To get yourself familiar with LUNA's functionality, we designed a few tasks. Please attempt to perform the following tasks and answer the questions below. For these tasks specifically, you will need to scan LUNA itself. You can download the LUNA project from here: <https://github.com/royvandijk06/luna> (↓ zip)

Please refer to the previous page for the instructions.

Task 1
Use LUNA to find what library or libraries used by LUNA are affected by a security vulnerability in the dependency 'estraverse@5.2.0'.

Task 2
Use LUNA to determine what files and functions inside LUNA's source code need to be changed when removing or replacing the library 'flast'.

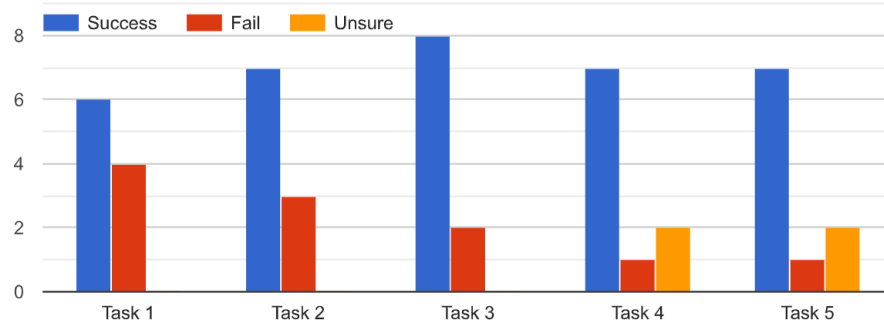
Task 3
Use LUNA to find which 5 API features of the internal library 'path' is being used by LUNA's code.

Task 4
Use LUNA to figure out the topological order in which the 7 js files inside the 'src' folder from LUNA use each other using a suitable layout, starting with *src/index.js*.

Task 5
You want to show a project collaborator a nice image of only the inner workings of *src/call-graph.js* from LUNA's source code. Use LUNA to do this.

Figure 4.1: The description of each task, that the interviewees had to perform.

Did you manage to succeed in each task with LUNA?



How difficult was each task to do with LUNA?

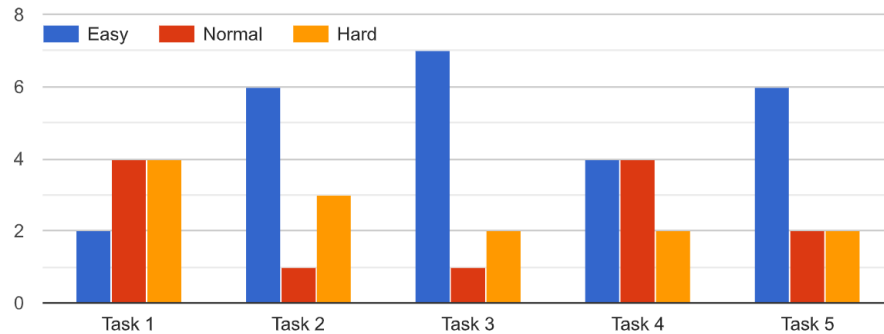


Figure 4.2: Some of the results of each task.

4.1 Research question 1 (RQ1)

The first research question asks how to recover the software architecture? In section 3.5, it was described how to perform source code analysis and what information can be extracted from the source code. This has been implemented in a tool called LUNA, which can do the following to recover the software architecture:

- Recover the scopes of the source code.
- Recover the classes and their methods.
- Recover the function calls and their definitions.
- Recover the directory and file structure.
- Recover the libraries used in the source code.
- Recover the imported API from the libraries.
- Recover the dependency relation between the libraries.
- Recover the dependency relation between the files.

As explained in chapter 3, the method to recover the software architecture is based on the source code analysis. An abstract syntax tree (AST) was constructed from the source code to extract the above information, by visiting each node from the AST and analyze itself, its scope and its references.

With the ten conducted interviews (appendix C), we can verify whether LUNA was successful in extracting the software architecture. Users had to perform several tasks testing different attributes of the architecture visualization.

Task 1 was added to test the library dependency relations in the architecture. It basically asked the user to figure out all connected dependencies for a specific library. Only 6 out of 10 users managed to complete this task, as they found it relatively difficult. The main problem with this task was a limitation by LUNA, where the search function does not consider collapsed nodes. So, when a node is collapsed, it will not search for nodes within this node. This caused many to not find the requested library. Future work (section 5.4) includes a suggestion to solve this issue.

Task 4 relates to the structure of the architecture and asks users to select a specific layout algorithm to position the architectural elements, i.e., the nodes in the graph, in such a way that a topological order becomes clear. Three people failed to do this, as two of them failed to use LUNA entirely due to technical issues and one of them did not know what to do. This task was determined to be moderately difficult.

The last task, task 5, asks users to create an image of the inner-architecture of a specific component within the project, this includes the function calls and their definitions, as well as the classes and their methods. Not only does this test users to use the screenshot feature of LUNA, it also tests them to identify the correct architecture and hide all irrelevant parts. Again, seven users were successful in this task and two did not manage to run LUNA. Yet, this task was considered to be fairly easy to do.

Hence, we may consider that LUNA is successfully able to recover the architecture of software.

4.2 Research question 2 (RQ2)

The second research question asks how to detect libraries and their usage? The answer to the previous research question (RQ1) already described how to recover the software architecture, which includes the libraries used in the source code. It does this by looking at the metadata of the project, as well as any library that is imported into the source code. RQ1's answer also mentioned that it recovers the API imported from the libraries, which means any functions, objects or data that is being offered by the library to the source code. Furthermore, the analysis of the AST also tracks where this API is being used in the source code. This means that the libraries and their usage can be detected by LUNA.

To confirm that this is indeed true, a few tasks were performed during the interviews (appendix C) to test the ability to understand library usage. Specifically, this was task 2 and task 3.

Task 2 asked users to find the affected files and functions when replacing or removing a used library. Seven out of ten users were able to complete task 2 successfully. Two users were not able to use LUNA at all, and one user encountered a usability issue, where they were unable to see the connection between the library and the files. Most users found this task to be easy to complete.

Task 3 asked users to find the library API used for a specific library. With this task, all users that were able to run LUNA successfully completed it with no effort.

4.3 Research question 3 (RQ3)

Research question 3 asks how to visualize the information in a useful way? The report LUNA generates includes a visualization about all the collected information. We want to know about the usability of this visualization. In the 10 conducted interviews (appendix C), a SUS test is performed. The System Usability Scale (SUS) [32] a reliable, low-cost usability scale that can be

used for global assessments of systems usability. The results of the SUS test are shown in table 4.1.

Item	Statement	Score
S1	I think that I would like to use LUNA frequently.	16
S2	I found LUNA unnecessarily complex.	26
S3	I thought LUNA was easy to use.	26
S4	I think that I would need the support of a technical person to be able to use LUNA.	30
S5	I found the various functions in LUNA were well integrated.	27
S6	I thought there was too much inconsistency in LUNA.	32
S7	I would imagine that most people would learn to use LUNA very quickly.	28
S8	I found LUNA very cumbersome to use.	25
S9	I felt very confident using LUNA.	22
S10	I needed to learn a lot of things before I could get going with LUNA.	23
Total		255
Overall score (0–100%)		63.75

Table 4.1: The combined SUS test results.

To calculate the overall SUS score, first the score contributions from each item has to be summed. Each item’s score contribution will range from 0 to 4. For items 1, 3, 5, 7, and 9 the score contribution is the scale position minus 1. For items 2, 4, 6, 8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU. SUS scores have a range of 0 to 100. Because the results of 10 interviews are used, everything will be 10x. The following mathematical formula (4.1) can be used to calculate the SUS overall score:

$$\text{SUS overall score} = 2.5 * \left(\sum_{i=1}^n (S1_i + S3_i + S5_i + S7_i + S9_i) - n * 1 + n * 5 - \sum_{i=1}^n (S2_i + S4_i + S6_i + S8_i + S10_i) \right) \quad (4.1)$$

Here, n would be 10 for the ten interviews that were performed and got the SUS results from.

The system has been tested by people with varying knowledge about programming and JavaScript. If we look at the results of the SUS test in figure 4.1, we can see that the overall score is 63.75%. This is above average, which means that the system is usable.

4.4 Main research question (RQ0)

By answering the previous research questions, we can answer the main research question. The main research question is: How to facilitate the comprehension of library usage in a software architecture?

The answer to this research question, LUNA was created. This tool can be used to recover the software architecture, detect the libraries and their usage, and visualize the information in a useful way, as we saw from our previous research questions. Visualization was chosen as the best way to convey information about library usage in software. In the interviews (appendix C), we see that from the people that may use LUNA in the future, they will use it to facilitate their comprehension about library usage in the projects:

- “To see how different modules [libraries] are correlated, and seeing which parts of my code use a certain library/method when refactoring”
- “I maintain a number of increasingly-complex Node.js library modules and getting a visual indication of how the inner workings interact can be useful.”

- “...als je code aanpast kun je de dependencies vrij makkelijk vinden. [...when you alter your code you can pretty easily find the dependencies]”
- “To see how different modules are correlated, and seeing which parts of my code use a certain library/method when refactoring”
- “...to verify how the upgrade of library impacts my own code.”

It is important to note that these answers were not from a leading question, but rather an open one, asking about their purpose for using LUNA. This means that these answers are not biased, as the people that were interviewed were not asked to answer in a certain way, but rather to answer honestly.

Chapter 5

Discussion

What is the relationship between this thesis and other work in the field? What are the implications and consequences of our work? Is there any threat to the validity? And what does future work look like? In this chapter, we will go through our findings carefully and address the above questions.

5.1 Comparison with Similar Projects

In chapter 3, it is explained why LUNA is developed to answer our research questions and how this tool works. Other tools exist that perform comparable functions or achieve similar aims. Let us examine those that were discovered during discovery and are of interest. We then compare them each to LUNA, the tool developed for this thesis. We first look at CodeGraph and HUNTER, the two projects most similar to LUNA, and then we explore other interesting projects that use different techniques to achieve a similar goal. Additionally, they all support JavaScript. In figure 5.1 an overview of different features across all the similar projects can be found.

Feature	LUNA	CodeGraph	HUNTER	Eunice	NPMGraph	JSCity	MetropolJS	js2flowchart	Source Code Explorer
JavaScript support	✓	✓	✓	✓	✓	✓	✓	✓	✓
Other language(s) support				✓					✓
Library dependency analysis	✓	✓	✓	✓	✓				
File dependency analysis	✓	✓	✓	✓					
Function analysis	✓		✓	✓		✓	✓	✓	✓
Class analysis	✓	✓	✓	✓				✓	
Library API analysis	✓			✓				✓	
File browser	✓	✓	✓						✓
Source code viewer	✓		✓					✓	✓
Graph visualization	✓	✓	✓		✓				
Treemap visualization			✓	✓		✓	✓		
Interaction with visualization	✓	✓	✓	✓	✓	✓	✓		✓

Table 5.1: Comparison between LUNA and similar tools.

5.1.1 CodeGraph

In Robert van Barlingen’s master thesis [33] CodeGraph was developed, which is a “web application that visualizes the dependency graph of JavaScript projects. This allows users to visually explore the relation between the different files in the project. Additionally, it provides the user with statistics on the project and information about the individual files. Through some usability tests, it was determined that CodeGraph facilitates better understanding of a JavaScript project and allows for a more pleasant user experience than traditional text-based tools.” A picture of CodeGraph can be found in figure 5.1.

When comparing CodeGraph with LUNA we see that both tools are similar in their goal and in the way they achieve this goal. Both tools visualize the dependency graph of a JavaScript project. However, there are some differences between the two tools. CodeGraph is a commercial project, while LUNA is fully open-source and accessible to anyone using node.js. One could say CodeGraph is a bit more advanced by offering more settings and different features. Such as, CodeGraph offers more layouts and layout customization, shows additional statistics, has git versioning integration, a minimap for the graph, and additional filtering options. However, compared to LUNA, it lacks node collapsing and expanding capabilities, color-coding in the graph, dynamic node sizes (based on file code size), and of course the detection of the inner architecture of the source code through function call-graphs and library API extraction. Furthermore, it is apparent by the presentation of LUNA that the focus lies on showing the usage of libraries. This is not the case for CodeGraph. CodeGraph appears to be a remarkably solid tool and a serious alternative to LUNA in most cases.

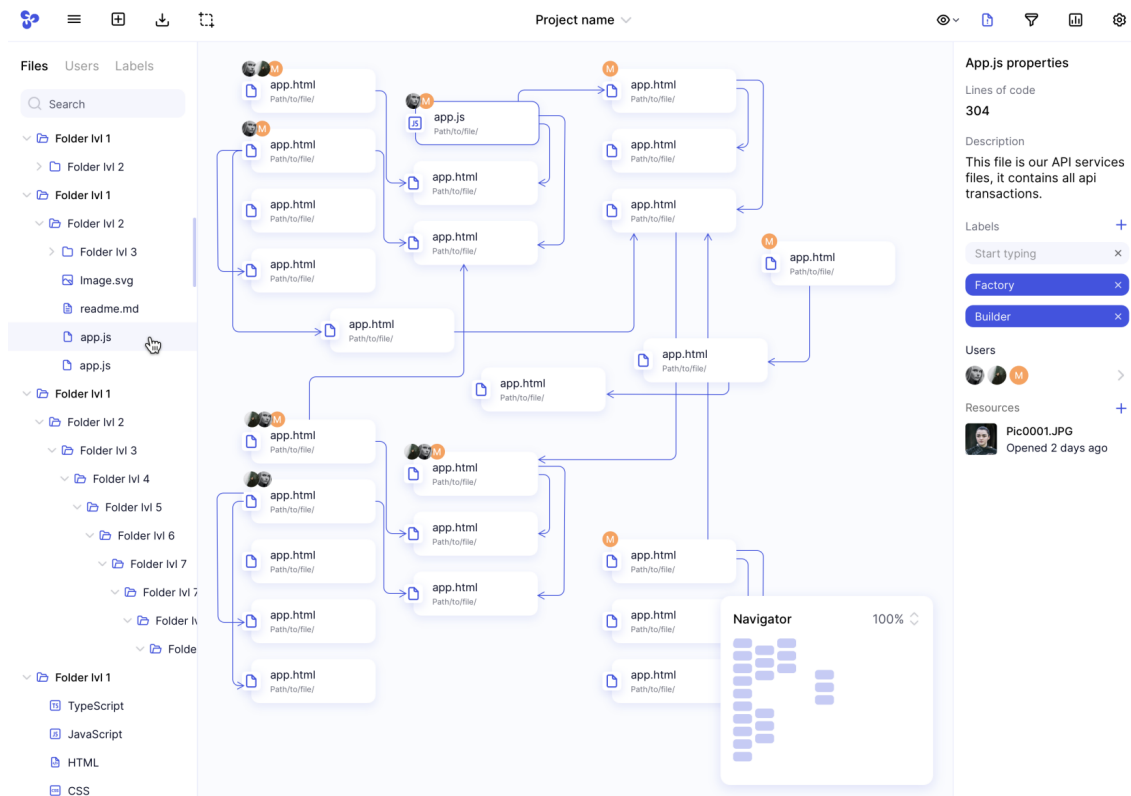


Figure 5.1: Overview of CodeGraph.

5.1.2 HUNTER

Hunter [34] is a “tool for the visualization of JavaScript applications. Hunter visualizes source code through a set of coordinated views that include a node-link diagram that depicts the dependencies among the components of a system, and a treemap that helps programmers to orientate when navigating its structure.” In figure 5.2, the GUI of the Hunter tool is shown, and its components are marked.

Again, the depiction of the dependency graph serves as Hunter’s key component. One of Hunter’s novel features is the color-coding of the folders in the file browser, which is not unlike the color-coding of LUNA. The color of the parent folder of the file that each node represents is used to identify each node in the dependency graph. This enables the user to visually connect the dependency graph visualization and the file browser’s view. In addition, each node’s size in the dependency graph is inversely correlated with the amount of lines of code that are present in the file. LUNA also does this for its nodes that represent files. This increases the visual weight of more significant nodes, or nodes with more lines of code. While LUNA has its function call-graphs, interestingly Hunter took a different approach by displaying a treemap of the available functions in the file and allow the user to inspect these functions in detail. Compared to LUNA, it loses the ability to connect functions through their invocations, but it gains the ability to observe the structure and nesting of the functions. This is a trade-off that is worth considering. We will actually see later in subsection 5.1.6 that there exists a way to achieve both in a single visualization.

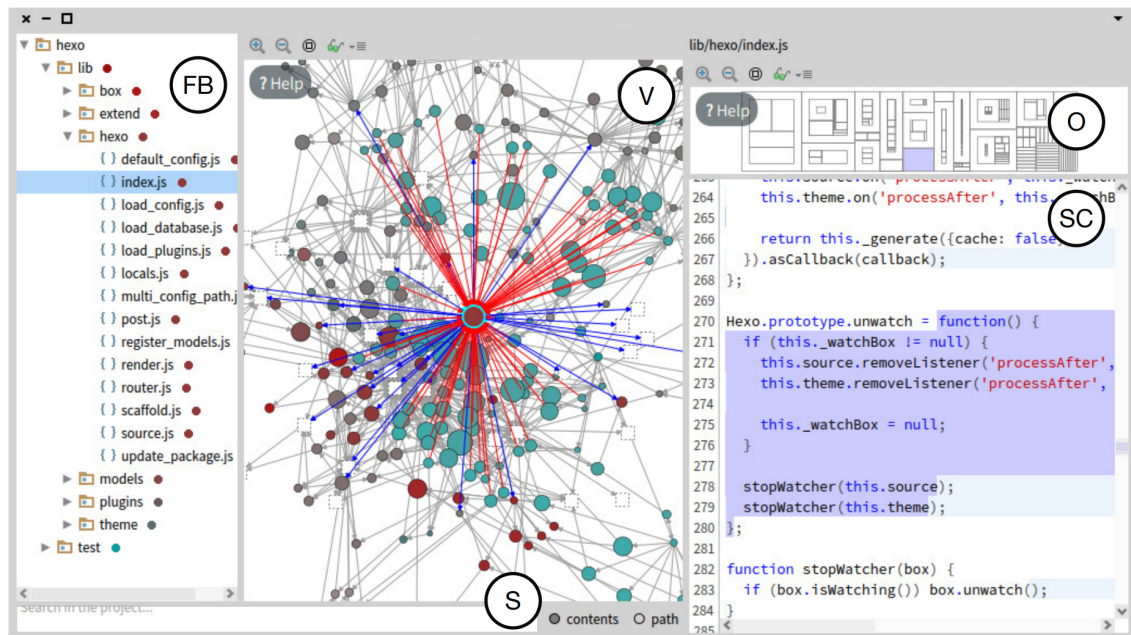


Figure 5.2: The GUI of Hunter. The left-most panel (FB) is a File Browser, also included in most IDEs. The center panel is the File Dependencies View (V), which relates dependencies between JavaScript source code files. The top-right panel (O) shows the structure in terms of functions nesting of the selected file. The bottom-left panel (S) is a Search box that can find specific files or functions located in V. Lastly, the bottom-right panel (SC) shows the Source Code of the selected file with the selected function highlighted.

5.1.3 Eunice

Eunice [35] is a tool that can scan C# and JavaScript projects and it “improves cohesion, coupling and modularity in software through hierarchical structure and simplified unidirectional dependencies. Eunice analyzes source code, infers its structure and shows if the dependencies match.” In

figure 5.3, the Eunice report of Eunice itself is shown.

Similarly to LUNA, a single command is needed to execute Eunice on JavaScript projects, namely `npx eunice`. Then, it creates an HTML report with an included visualization. However, it is the visualization and the information it displays that is very different compared to LUNA. Eunice visualizes a nested tree-like structure, whereas LUNA visualizes a collapsible/expandable graph network. Admittedly, Eunice is impressive and information-dense, but also quite complex and has a steep learning curve.

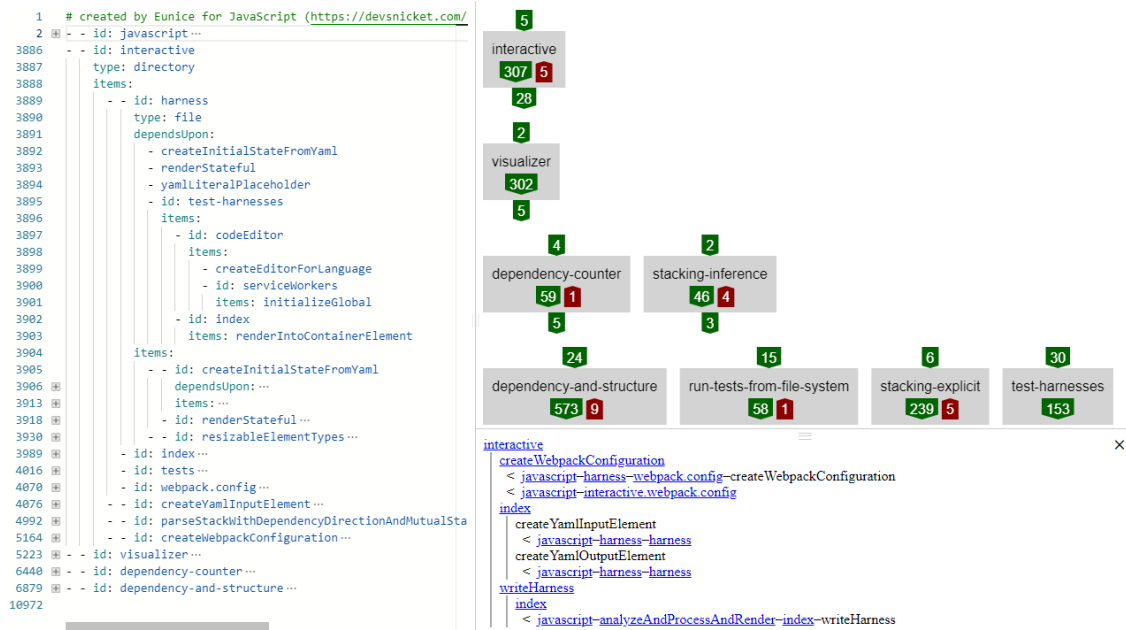


Figure 5.3: Eunice’s HTML report.

5.1.4 NPMGraph

NPMGraph [36] is a web application that has a simple visualization of the dependency graph of a JavaScript project. It is available through the worldwide web¹. In figure 5.4, NPMGraph is demonstrated.

LUNA includes most of the functionality provided by NPMGraph. The graph generated by NPMGraph is mostly static, but it allows for some interactability, such as selecting nodes or collapsing nodes with its children. This is of course similar to LUNA. Yet NPMGraph shows some additional information, such as the bundle size NPMS.io score and the list of maintainers. LUNA colorcodes by libraries, but with NPMGraph you can choose the colorization of your preference (NPMS.io overall score, NPMS.io quality score, NPMS.io popularity score, NPMS.io maintenance score, or # of maintainers). One of the best features of NPMGraph, is that you can choose any of the displayed libraries in the graph for NPMGraph to scan and make a new graph in this library’s context. This is something I tried to replicate with LUNA, but due to limitations it can only show a command line that can rerun LUNA on the selected library (See G from figure 3.12). NPMGraph is a simple tool that is easy to use and is a good alternative to LUNA when you are only interested in the dependency graph of a project.

¹<https://npmgraph.js.org/>

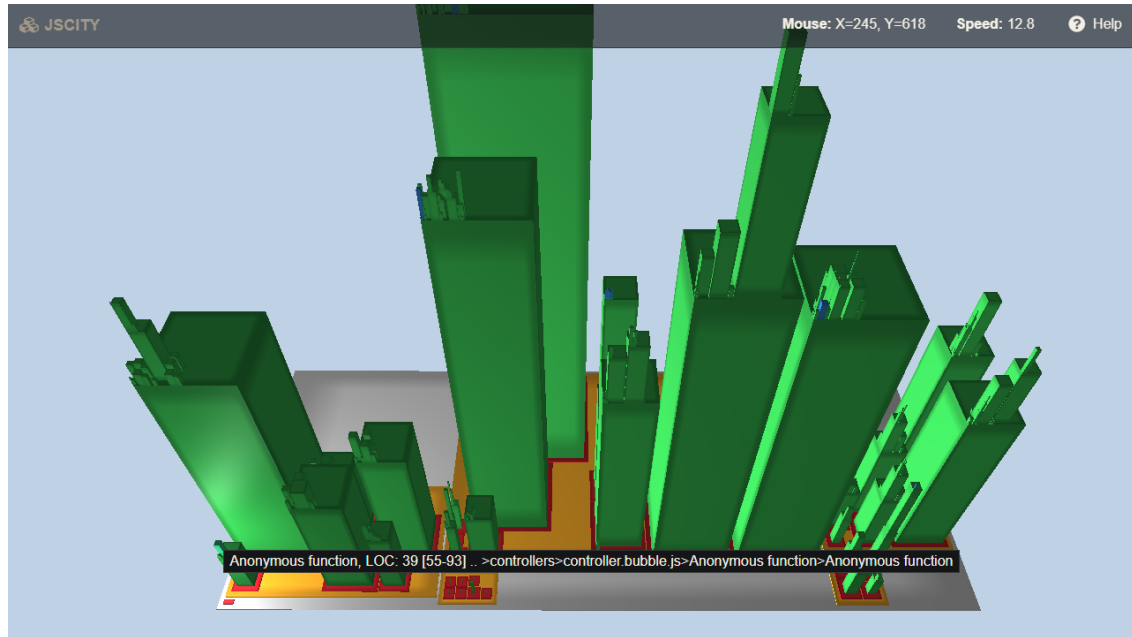


Figure 5.5: Screenshot of JSCity.

5.1.6 MetropolJS

MetropolJS [38] visualizes and debugs large-scale JavaScript code structure with treemaps. This tool aims to solve the following problem: “As a result of the large scale and diverse composition of modern compiled JavaScript applications, comprehending overall program structure for debugging proves difficult.” Therefore, MetropolJS provides “an optimized approach for visualizing complex program structure that enables new debugging techniques where the execution of programs can be displayed in real time from a bird’s-eye view. The approach facilitates highlighting and visualizing method calls and distinctive code patterns on top of code segments without a high overhead for navigation. Using this approach enables fast analysis of previously difficult-to-comprehend code bases.” Figure 5.6 shows a screenshot of MetropolJS in action.

The function call-graphs of LUNA show all the possible function calls in a project, but they do not show the nesting of the functions. MetropolJS shows the nesting of the functions with colored information, while also showing the function calls. Not only that, but it animates the function calling order. This is a very interesting approach that I have not seen before. It is a very good alternative to LUNA when you are interested in the nesting of the functions. However, it does not show the dependencies between the files and libraries, which is a feature that LUNA has.



Figure 5.6: MetropolJS in action

5.1.7 js2flowchart.js

The library `js2flowchart.js` [39] is a visualization tool for generating SVG-based flowcharts from JavaScript source-code. It helps to explain or document your code via flowcharts. `Js2flowchart.js` works by defining abstraction levels to render only import/exports, classes/function names, function dependencies to learn/explain the code step by step. It also has a presentation generator to generate a list of SVGs in order to different abstractions levels. It also has defined flow tree modifiers to map well-known JavaScript native APIs. Furthermore, it also contains a destruction modifier to replace a block of code with a single shape on scheme. Additionally, you can design your own custom flow tree modifiers using this. In addition, it contains a flow tree ignore filter to totally omit specific code nodes, such as log lines. Also, it contains a focus node or a whole code logic branch to draw attention to key portions of the scheme. To conceal less-important information, it also has a blur node or an entire code logic branch. It also offers specified styles, themes, and support; pick the one you prefer. Moreover, it supports style customization and offers a convenient API for changing particular styles without using boilerplate. Figure 5.7 shows

a screenshot of js2flowchart.js in action.

Compared to LUNA, it does not have much in common. However, the idea of visualizing the code as a flowchart is interesting. It is a good alternative to LUNA when you are only interested in visualizing the flow of your code.

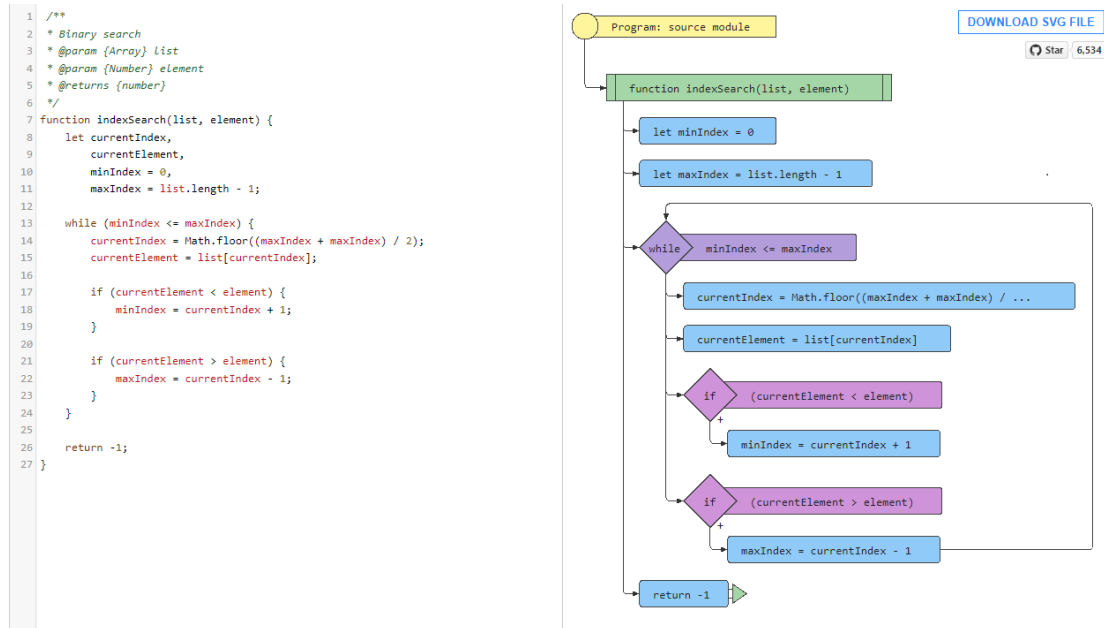


Figure 5.7: Demonstration of js2flowchart.js

5.1.8 JSClassFinder

In the early days of JavaScript, especially before ES6 introduced class support, JavaScript was dominated by class-like structures. Even nowadays they are still used. JSClassFinder [40] is a tool that can detect these class-like structures. Likewise to LUNA, it scans the AST to achieve this. Furthermore, with the power of the Moose² framework, it can produce powerful visualizations, such as the one in figure 5.8.

²<http://moosetechnology.org/>

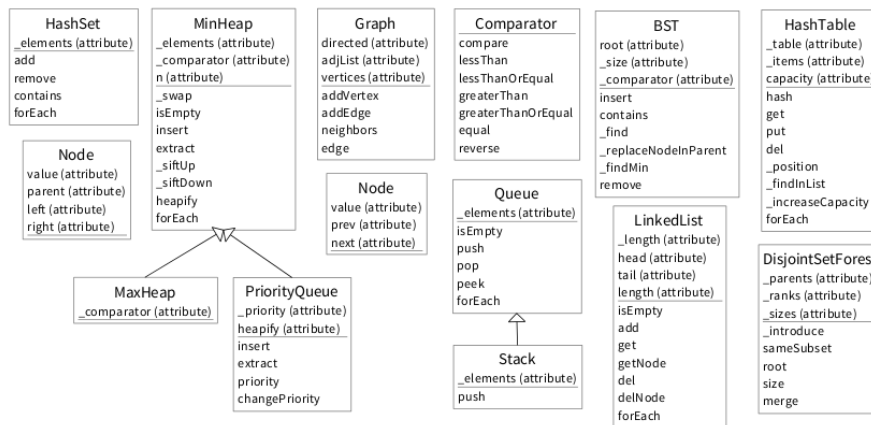


Figure 5.8: A Class Diagram Generated by JSClassFinder (for JSClassFinder's algorithm.js)

5.1.9 Source Code Explorer

During my studies I have actually helped develop a system before that shares similar goals to what LUNA tries to achieve. We called it Source Code Explorer and was made for a visualization course. It is accessible through the worldwide web³. In figure 5.9, Source Code Explorer is demonstrated.

While LUNA tries to abstract the architecture of the source code of a project, Source Code Explorer tries to visualize the source code itself. It also lacks any ability to show the dependencies between the files and libraries. However, it has limited capabilities to show the nesting of the functions. It is a good alternative to LUNA when you are interested in the source code itself, but it is not a good alternative when you are interested in the architecture of the source code or the role of the libraries within the project.

³<https://src-explorer.glitch.me>

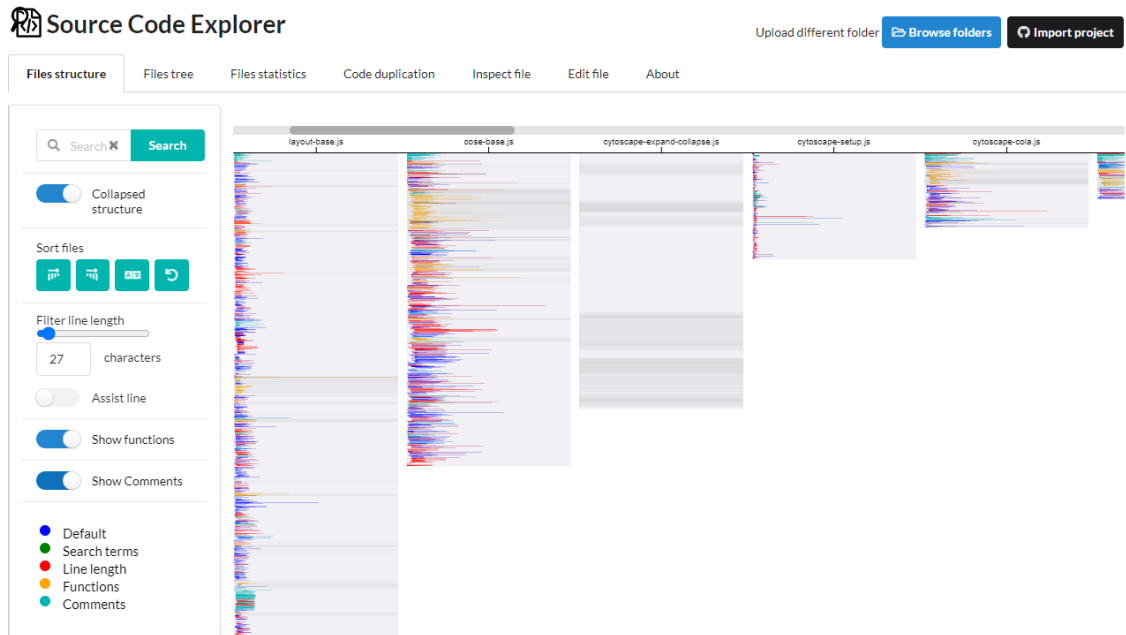


Figure 5.9: The GUI of the Source Code Explorer

5.2 Implications

We summarize our implications with the following takeaway messages for the key stakeholders:

5.2.1 Researchers

Researchers can use LUNA to analyze the usage of libraries in JavaScript software, which can be used to answer research questions relating to the usage of libraries in JavaScript software. For this, LUNA provides a unique ability to research the connection between software architecture and libraries. It also serves as an example how to analyze a dynamically typed language, like JavaScript, as this is a difficult and non-trivial task. Furthermore, they can take LUNA as a starting point to develop their own tool to analyze the usage of libraries in JavaScript software (at scale). Alternatively, they may extend LUNA with additional features matching their research. This is possible, as LUNA is fully open-source [41]. Some ideas on how to update and extend LUNA can be found in section 5.4 about future work. Finally, researchers may compare LUNA against other tools that analyze the usage of libraries in software, or tools that visualize software architecture, in order to find the best performant according to some attributes.

5.2.2 Practitioners

In the context of this thesis, practitioners are software engineers and maintainers that use libraries in their software. They can apply LUNA for various use-cases. They can use LUNA to help evaluate the impact a library has on their project, e.g. when breaking changes are introduced when updating it. Moreover, they may use LUNA to realize the complexity of their project by using libraries that provide simplification or abstraction. They can also use LUNA's visualization to showcase information about libraries usage to other interested parties. For example, imagine you want to extend or improve an open-source project, and you decide to make a pull request (PR) for it. LUNA's visualization would be a good way to familiarize yourself with its codebase and architecture, instead of looking through code and trying to find where the relevant code is. Not only information about libraries, but also about the project's architecture can be shown to

others. For example, use LUNA to put its generated visualization in the project's README.md, i.e., its description. Lastly, LUNA can be used by practitioners to find out what and where exactly in the project code needs to be changed when replacing or removing used libraries.

5.3 Threats to validity

In this section, we look at the threats to validity of our research. We discuss the following groups of validity: construct validity, internal validity and external validity.

5.3.1 Construct

Construct validity is the extent to which the measurements in the study reflect real-world situations, and the extent to which the measurements in the study measure what they are supposed to measure. In our case, construct validity is about how well the tool measures the concept it was designed to evaluate, which is the usage of libraries in JavaScript software. We have shown in chapter 3 that LUNA is able to recover the software architecture. However, is this truly a valid representation of the software architecture? In LUNA's visualization we show the directory structure of the project and the included source code files with their inner-relations. For each file, a sub-graph can emerge that shows the chain of function calls and class relations. It can be argued that this is not a valid representation of the software architecture. However, it is up to interpretation.

5.3.2 Internal

Internal validity is related to uncontrolled aspects that may affect the experimental results. The expertise of interview participants might not be representative of a real-world sample of professional software developers. To mitigate this threat, questions (appendix C) were asked in the interview to assess their experience using JavaScript and libraries.

5.3.3 External

External validity is related to the possibility to generalize our results. First and foremost, we have to consider that the results of this thesis are only valid for JavaScript software. However, I argue that the usage of libraries in JavaScript software is similar to the usage of libraries in most other software. Therefore, we may assume that the results of this thesis are also valid for most other software.

Furthermore, we have to consider that the results of this thesis are only valid for the projects that we have tested. The chosen open-source JavaScript projects used for testing may not represent the entirety of JavaScript software well enough. To mitigate this risk, a wide range of projects were carefully chosen to maximize coverage.

Only 10 people were interviewed. They may not represent the entire developer community well enough. More interviews could have been conducted to get a better understanding of the usability of LUNA.

5.4 Future work

In this section, we discuss all the directions I would like to take LUNA in the future. A lot of these were suggested by our test participants that were mentioned in chapter 4. We will discuss the different directions separately.

5.4.1 Scanner

The scanner component of LUNA might be greatly improved. It currently lacks export tracking for files, which means API detection is solely based on import statements. Export tracking would allow us to detect the full available API of a file and can even be extended to whole libraries.

Currently, the scanner only saves the position of function definitions. However, it would be more useful for humans to also save the line number and even the column number. This would allow us to show the exact location of a function in the source code.

Also, additional metrics and statistics could be extracted by the scanner. More information about the relation between the scanned project and external components could be useful. For example, the proportion of lines that depend on an external component, either directly or indirectly. Or the proportion of lines that are used from an external component. Not only for lines, but also for functions.

To construct the architecture of a file, the scanner currently only looks at function calls with limited class structure support. However, it would be useful to provide more meaningful abstractions for classes and objects in the code. This would allow us to show the architecture of a file in more detail.

It relies on the file system to perform scanning. It would help with accessibility, if LUNA could be used fully online without having to install anything on your system. Although the installation requirement is already pretty minimal, it still is a barrier of entry to try out LUNA. It could use platforms like GitHub⁴, GitLab⁵ or BitBucket⁶ to import repository to scan. Although definitely possible, the challenge lies in rewriting a lot of the codebase to handle this change.

The scanner currently only supports JavaScript source code. However, it would be useful to support other programming languages or runtimes as well. TypeScript, JSX, HTML, and even less related programming languages such as Java or Python are examples. This would allow us to scan more projects and provide more information about the usage of libraries in software.

LUNA can currently only scan one project at a time in order to create a visualization. Instead of generating a visualization, it could scan multiple projects at once. This would allow us to perform quantitative research on the usage of libraries in software.

Finally, the scanner's handling of package versions is very basic. It only looks at the version number and not at the version range. Meaning that constructing dependency trees might not be accurate. This could be improved by considering the version range as well.

5.4.2 Visualization

The visualization component of LUNA can be improved in many ways. First, the visualization currently uses one global layout. However, it would be useful to split the layout into multiple layouts for each subcomponent. This would allow us to show more information about the architecture of a file.

Secondly, the visualization currently allows overlapping compound nodes for most layout algorithms. However, this can be confusing and unappealing for the user. It would be useful to avoid overlapping compound nodes for all layouts. Sadly, to my knowledge, this is simply not possible with the current implemented layout algorithms. So, this would require altering the layout algorithms themselves.

Furthermore, only the left mouse button is used to interact with the graph: both to drag nodes around and to pan the graph by dragging the background. This can be annoying for the user when they want to pan the graph but accidentally drag a node. It would be useful to use the right mouse button for panning and the left mouse button for dragging nodes. This would allow us to interact with the graph more intuitively. To my knowledge, this is not possible with the cytoscape.js library [27] that handles this. There exists an issue on GitHub⁷ that suggests this,

⁴<https://github.com/>

⁵<https://gitlab.com/>

⁶<https://bitbucket.org/>

⁷<https://github.com/cytoscape/cytoscape.js/issues/3063>

but it has not been implemented yet.

Another idea that could improve the visualization is to change the edges depending on the closeness to the node that is currently selected. For example, the thickness or opacity of the edges could be changed. This would allow us to show the user which edges are more important/related to the node than others.

Also, the graph currently shows all edges between visible nodes. However, this can be confusing or overwhelming for the user. It would be useful to implement techniques that can reduce the amount of edges shown. For example, Tarjan's Algorithm⁸ could be used to reduce the amount of edges shown. This would allow us to show the user only the most important edges.

Currently, the cola layout algorithm can cause some inconvenience for the user. It runs a simulation to produce a layout. However, this simulation is currently set to run at most 4 seconds and while it is running, the user cannot interact with the graph without being interrupted by the simulation. An improvement could be made by allowing the user to interact with the graph while the simulation is running without being interrupted. This would allow us to interact with the graph more intuitively.

The cytoscape.js library [27] that I use to construct and visualize the graph comes with an option to show a visual indicator when a node can be expanded or collapsed. However, during optimization of the visualization, I removed this indicator, because on-demand loading of nodes made it uncertain when a node could be expanded or collapsed. However, a way could be found to bring this indicator back to the visualization. This would allow users to know when a node can be expanded or collapsed before trying. This indicator also provides the user an alternative (perhaps more intuitive) way to expand or collapse a node.

5.4.3 User Interface

There are several improvements that can be made to the user interface of LUNA. This mostly concerns the menu and the graph, but some improvements touch upon the user interface of visualization as well.

In the menu, there is currently no search box for the library and the file tree sections. However, it would be convenient to have a search box there, as these can be very large. This would allow users to search for a specific item more easily.

The center button of the node menu is currently used to locate the selected node in the graph. However, it would possibly be useful to highlight the selected node as well, as this would allow users to see which node is selected more easily.

Currently, the search button only works for visible nodes. However, it would be great to make it search for collapsed items as well. This would allow users to search for items that are not visible, but do exist in the graph. When a user searches for a node that is collapsed, the node could be expanded automatically. Or, when a user searches for a node that is collapsed, the user could be notified that the node is collapsed.

Currently, the hiding status of a parent item in the menu is not linked to the hiding status of its children items. However, it would be useful to link these statuses. This would allow users to hide all items in a section with one click.

Additionally, it would be great to have an option to hide all other items, besides those selected. This would allow users to focus solely on the items they are interested in.

Likewise, it would be useful to add the ability to collapse or expand all nodes in the graph. This would allow users to quickly collapse or expand all nodes in the graph.

Right now all node options and information is located in the node menu. But for some user it may be more convenient to have these options in a context menu, that can be activated with right-click. This would allow these users to interact with the graph more intuitively.

The visualization currently shows all functions inside a file. However, may not want to see all functions. Filter options could be added to the user interface. This would allow users to filter the functions and focus on the functions they are interested in.

⁸https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm

Finally, a light theme option could be added to the user interface for those who prefer a light theme over a dark theme. This would allow those users to use LUNA more comfortably.

5.4.4 Bugs

LUNA has a few known issues. One major one is caused by the library used to handle collapsing and expanding of nodes: `cytoscape-expand-collapse` [28]. Sometimes nodes do not expand or collapse properly. This may result in LUNA breaking. Best to avoid excessive collapsing and expanding of nodes until a solution for this issue is created.

Lastly, there are some browser related issues. LUNA is currently only fully supported by chromium based web browsers, like Google Chrome. Other browsers may have issues with the visualization.

5.4.5 Beyond LUNA

As noted in the use cases for researchers, LUNA may be compared against other tools that examine the usage of libraries in software or tools that display software architecture to determine which is the most performant based on particular parameters.

Furthermore, the interviews obtained may be reused in research that is not associated to LUNA, such as studies on the usability of software visualization tools. The interviews may be used to discover what users expect from a software visualization tool, as well as what they find beneficial and puzzling. This might be used to improve the overall usability of software visualization tools.

Chapter 6

Conclusion

In conclusion, the objective of this thesis is to illustrate how software packages are employed in software architecture, in order to help with developers' comprehension. This is accomplished through investigation into how software libraries are used in software architecture. We discussed the problem's relevance and identified the stakeholders. Following that, we went over the research questions and explained why they are important to the research problem. We presented related work, study methodology, and study results.

Our approach was to create a tool called LUNA. This tool works by scanning a JavaScript project, constructing an AST from the source code, and extracting information about the software architecture and library usage from it. This data is then visualized in a web report. The visualization has three sub-graphs and an interaction panel. The first sub-graph depicts the project's directory and file structure, and each file contains an abstraction represented by a function call-graph. The second sub-graph displays the libraries used in the project, which are divided into internal, external, and unused/development libraries. The third sub-graph reveals the libraries' dependencies. Relations are indicated within each sub-graph and between sub-graphs. The interaction panel allows the user to interact with the visualization.

The research results show that LUNA is able to recover the software architecture and detect libraries and their usage. The results have also shown that we can visualize this information in a useful way. The results have been compared with related work, and the implications in the field have been discussed. At last, the threats to validity were debated.

Bibliography

- [1] Wikipedia contributors, “Software package — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=Software_package&oldid=1104525135, 2022. [Online; accessed 18-October-2022]. 1
- [2] Wikipedia contributors, “Software architecture — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=Software_architecture&oldid=1115479169, 2022. [Online; accessed 27-October-2022]. 5
- [3] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. SEI Series in Software Engineering, Boston, MA: Addison Wesley, 4 ed., Oct. 2021. 5, 6
- [4] R. Agarwal, R. Deshmukh, P. Borhade, S. Murarka, and D. Datta, “Software architecture recovery techniques,” *International Journal of Engineering and Advanced Technology*, vol. 9, p. 4, 04 2020. 5, 7
- [5] J. Garcia, I. Ivkovic, and N. Medvidovic, “A comparative analysis of software architecture recovery techniques,” in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 486–496, IEEE, 2013. 5, 7
- [6] R. M. Tarawaneh, P. Keller, and A. Ebert, “A General Introduction To Graph Visualization Techniques,” in *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011* (C. Garth, A. Middel, and H. Hagen, eds.), vol. 27 of *OpenAccess Series in Informatics (OASICs)*, (Dagstuhl, Germany), pp. 151–164, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012. 6, 7
- [7] R. G. Kula, A. Ouni, D. M. German, and K. Inoue, “On the impact of micro-packages: An empirical study of the npm javascript ecosystem,” *arXiv preprint arXiv:1709.04638*, 2017. 6
- [8] A. Zerouali, T. Mens, G. Robles, and J. M. Gonzalez-Barahona, “On the diversity of software package popularity metrics: An empirical study of npm,” in *2019 IEEE 26th international conference on software analysis, Evolution and Reengineering (SANER)*, pp. 589–593, IEEE, 2019. 6
- [9] J. Žitný, “Npm a javascript,” Master’s thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2017. 6
- [10] C. Bautista, A. Dahiya, K. Hardgrave, and D. Xu, “Javascript documentation generation through semantic code analysis,” 6
- [11] B. Taraghi, A. Azmi, and O. Yusop, “Producing software engineering documents through software reverse engineering for node.js web application,” in *Advanced Research in Engineering and Information Technology International Conference (AVAREIT)*, 2018. 6
- [12] A. Nurwidiantoro, T. Ho-Quang, and M. R. V. Chaudron, “Automated classification of class role-stereotypes via machine learning,” in *Proceedings of the Evaluation and Assessment on Software Engineering, EASE ’19*, (New York, NY, USA), p. 79–88, Association for Computing Machinery, 2019. 7

- [13] L. Xavier, A. Brito, A. Hora, and M. T. Valente, “Historical and impact analysis of api breaking changes: A large-scale study,” in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 138–147, IEEE, 2017. 7
- [14] L. Ochoa, T. Degueule, J.-R. Falleri, and J. Vinju, “Breaking bad? semantic versioning and impact of breaking changes in maven central,” *arXiv preprint arXiv:2110.07889*, 2021. 7
- [15] H. Mili, F. Mili, and A. Mili, “Reusing software: Issues and research directions,” *IEEE transactions on Software Engineering*, vol. 21, no. 6, pp. 528–562, 1995. 7
- [16] W. B. Frakes and P. Gandel, “Representing reusable software,” *Information and Software Technology*, vol. 32, no. 10, pp. 653–664, 1990. 7
- [17] W. B. Frakes and T. P. Pole, “An empirical study of representation methods for reusable software components,” *IEEE transactions on software engineering*, vol. 20, no. 8, pp. 617–630, 1994. 7
- [18] A. Feldthaus, M. Schäfer, M. Sridharan, J. Dolby, and F. Tip, “Efficient construction of approximate call graphs for javascript ide services,” in *2013 35th International Conference on Software Engineering (ICSE)*, pp. 752–761, IEEE, 2013. 7
- [19] B. B. Nielsen, M. T. Torp, and A. Møller, “Modular call graph construction for security scanning of node.js applications,” in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 29–41, 2021. 7
- [20] acornjs, “Acorn.” <https://github.com/acornjs/acorn>, 2021. 11
- [21] R. Kooi, “scope-analyzer.” <https://github.com/goto-bus-stop/scope-analyzer>, 2021. 11
- [22] PerimeterX, “flAST - FLat Abstract Syntax Tree.” <https://github.com/PerimeterX/flast>, 2022. 11
- [23] ESLint, “ESLint Scope.” <https://github.com/eslint/eslint-scope>, 2022. 11
- [24] ESLint, “Espree.” <https://github.com/eslint/espree>, 2022. 11, 55
- [25] npm, “npm - a JavaScript package manager.” <https://github.com/npm/cli>, 2010. 12, 21
- [26] M. Eernisse, “EJS – Embedded JavaScript templates.” <https://ejs.co/>, 2016. 13
- [27] M. Franz, C. T. Lopes, G. Huck, Y. Dong, O. Sumer, and G. D. Bader, “Cytoscape.js: a graph theory library for visualisation and analysis,” *Bioinformatics*, vol. 32, pp. 309–311, 09 2015. 13, 40, 41
- [28] U. Dogrusoz, A. Karacelik, I. Safarli, H. Balci, L. Dervishi, and M. C. Siper, “Efficient methods and readily customizable libraries for managing complexity of large networks,” *PLoS one*, vol. 13, no. 5, p. e0197238, 2018. 13, 42
- [29] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo, “A technique for drawing directed graphs,” *IEEE Transactions on Software Engineering*, vol. 19, no. 3, pp. 214–230, 1993. 15
- [30] M. Jünger and P. Mutzel, “2-layer straightline crossing minimization: Performance of exact and heuristic algorithms,” in *Graph Algorithms And Applications I*, pp. 3–27, World Scientific, 2002. 15
- [31] T. jQuery Team, “jQuery.” <https://github.com/jquery/jquery>, 2006. 21
- [32] J. Brooke, “Sus: A quick and dirty usability scale,” *Usability Eval. Ind.*, vol. 189, 11 1995. 25

-
- [33] R. Van Barlingen, “CodeGraph: an Interactive Dependency Analyzer for JavaScript Projects,” master’s thesis, Aalto University. School of Science, 2021. 30
- [34] M. Dias, D. Orellana, S. Vidal, L. Merino, and A. Bergel, “Evaluating a visual approach for understanding javascript source code,” in *Proceedings of the 28th International Conference on Program Comprehension*, pp. 128–138, 07 2020. 31
- [35] G. Dyson, “Eunice.” <https://devsnicket.com/eunice/>, 2020. 31
- [36] npmgraph, “npmgraph - a tool for exploring npm modules and dependencies.” <https://npmgraph.js.org>, 2018. 32
- [37] M. Viana, A. Hora, and M. T. Valente, “Codecity for (and by) javascript,” *arXiv preprint arXiv:1705.05476*, 2017. 33
- [38] J. D. Scarsbrook, R. K. L. Ko, B. Rogers, and D. Bainbridge, “MetropolJS: Visualizing and Debugging Large-Scale JavaScript Program Structure with Treemaps,” in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pp. 389–3893, May 2018. 34
- [39] B. Liashenko, “js2flowchart - a visualization library to convert any JavaScript code into beautiful SVG flowchart..” <https://github.com/Bogdan-Lyashenko/js-code-to-svg-flowchart>, 2017. 35
- [40] L. H. Silva, D. Hovadick, M. T. Valente, A. Bergel, N. Anquetil, and A. Etien, “Jsclassfinder: A tool to detect class-like structures in javascript,” *arXiv preprint arXiv:1602.05891*, 2016. 36
- [41] R. van Dijk, “LUNA: Library Usage in Node.js Analyzer.” <https://github.com/royvandijk06/luna>, 2022. 38

Appendix A

MoSCoW

The project requirements for building LUNA used the MoSCoW method for prioritization. This method has four levels of prioritization, namely:

Must have The requirements with this priority are critical for the product. These requirements are the Minimum Usable Subset.

Should have These are important but not vital or fundamental to the product. There are often workarounds available.

Could have Desirable requirements but can be left out when time and resources are not available.

Won't have Requirements which will not be delivered in the time frame. Those are the least critical, but are in the list to clarify the scope of the project.

A.1 Constraints

UR1-1 LUNA is available as a NPM package	<i>M</i>
UR1-2 LUNA is available as a Node.js Package Runner	<i>M</i>
UR1-3 LUNA is available as a standalone application	<i>C</i>
UR1-4 LUNA reports can be viewed on chromium based web browsers (Chrome, Edge, Brave)	<i>M</i>
UR1-5 LUNA reports can be viewed on Firefox	<i>C</i>
UR1-6 LUNA reports can be viewed on other web browsers	<i>C</i>
UR1-7 LUNA is executable on Windows	<i>M</i>
UR1-8 LUNA is executable on Linux	<i>S</i>

UR1-9 LUNA is executable on Mac	<i>S</i>
UR1-10 LUNA is executable on Android	<i>W</i>
UR1-11 LUNA is executable on iOS	<i>W</i>

A.2 Project

UR2-1 LUNA can work with Node.js projects	<i>M</i>
UR2-2 LUNA can work with other JavaScript projects	<i>S</i>
UR2-3 LUNA can work with TypeScript projects	<i>C</i>
UR2-4 LUNA can work with JSX projects	<i>C</i>
UR2-5 LUNA can work with projects based on other programming languages	<i>C</i>
UR2-6 LUNA can scan local projects	<i>M</i>
UR2-7 LUNA can scan online project repositories (e.g from GitHub).	<i>C</i>
UR2-8 LUNA can scan projects with dependencies installed	<i>M</i>
UR2-9 LUNA can scan projects without dependencies installed	<i>M</i>
UR2-10 LUNA can scan projects at different times of development using a versioning system (e.g. git)	<i>W</i>
UR2-11 LUNA can show commit information using a versioning system (e.g. git)	<i>W</i>
UR2-12 LUNA can generate HTML reports that visualize library use in projects	<i>M</i>
UR2-13 LUNA can generate reports in other formats (e.g. PDF, XML)	<i>W</i>

UR2-14 *C*
LUNA scans can be configured using config files in the project

A.3 GUI

UR3-1 *M*
LUNA reports display a graph

UR3-2 *M*
LUNA reports display a menu for graph settings and actions

UR3-3 *M*
LUNA reports display a menu for node information and actions

UR3-4 *S*
Each menu option / action has a clear description for the user

A.4 Graph

UR4-1 *M*
The graph has a source code subgraph component

UR4-2 *M*
The graph has a libraries' subgraph component

UR4-3 *M*
The graph has a dependencies' subgraph component

UR4-4 *M*
Every JavaScript file in the project is represented as a node in the source code component

UR4-5 *W*
Every other file in the project is represented as a node in the source code component

UR4-6 *M*
Every imported library in the project is represented as a node in the libraries component

UR4-7 *M*
Every dependency of the libraries in the project is represented as a node in the dependency graph component

UR4-8 *M*
The source code component can include inner components representing the folder structure

UR4-9 *M*
The library component includes inner components representing library categories, e.g external or internal.

UR4-10	<i>M</i>
The dependency graph component shows a different color for the dependency tree of each library	
UR4-11	<i>S</i>
A file node may include a subgraph representing a function callgraph	
UR4-12	<i>S</i>
A library node may include a subgraph representing its exposed API	
UR4-13	<i>M</i>
A function calling another function or library API is represented by a line connecting two nodes	
UR4-14	<i>M</i>
If function A calls another function or library API B, then the line connecting nodes A and B has an arrow pointing from node A to node B	
UR4-15	<i>M</i>
A file importing another file or library is represented by a line connecting two nodes	
UR4-16	<i>M</i>
If file A imports file or library B, then the line connecting nodes A and B has an arrow pointing from node A to node B	
UR4-17	<i>M</i>
A library depending on another library or dependency is represented by a line connecting two nodes	
UR4-18	<i>M</i>
If library A depends on library B, then the line connecting nodes A and B has an arrow pointing from node A to node B	
UR4-19	<i>M</i>
A node displays the name of the file / function / API / library / dependency	
UR4-20	<i>S</i>
Users can choose between different layouts, which define where the nodes are placed	
UR4-21	<i>C</i>
The user can interactively change the position of the nodes, e.g by dragging them.	
UR4-22	<i>S</i>
The user have the ability to filter which files are shown in the graph	
UR4-23	<i>S</i>
The user have the ability to filter which libraries are shown in the graph	
UR4-24	<i>W</i>
The user should be able to view the graph in 3D	
UR4-25	<i>C</i>
The user can save a screenshot of the graph	

UR4-26	<i>S</i>
The user can zoom and pan the graph	

UR4-27	<i>S</i>
The user can reset zoom or position	

A.5 Node

UR5-1	<i>M</i>
A node in the graph can be highlighted	

UR5-2	<i>M</i>
A node in the graph can be selected by clicking or hovering	

UR5-3	<i>M</i>
A node in the graph may be collapsed or expanded, depending on if it has children nodes	

UR5-4	<i>M</i>
A node in the graph can be displayed in the menu	

UR5-5	<i>S</i>
A node in the menu can be highlighted	

UR5-6	<i>M</i>
A node in the menu displays information	

A.6 Information

UR6-1	<i>M</i>
A node representing a file shows its file name	

UR6-2	<i>M</i>
A node representing a file shows its disk location	

UR6-3	<i>M</i>
A node representing a file shows its source code size	

UR6-4	<i>S</i>
A node representing a file that can be opened, displaying its contents	

UR6-5	<i>M</i>
A node representing a library shows its library name	

UR6-6	<i>S</i>
A node representing a library shows its library version	

UR6-7	<i>S</i>
A node representing a library shows an external link to its corresponding NPM page	

UR6-8	<i>S</i>
A node representing a function call shows its function name	
UR6-9	<i>M</i>
A node representing a function call shows its position in the source code file	
UR6-10	<i>M</i>
A node representing API shows its function name	

A.7 Scalability

UR7-1	<i>M</i>
LUNA can handle projects containing only 1 JavaScript file	
UR7-2	<i>M</i>
LUNA can handle projects containing up to 10 JavaScript files	
UR7-3	<i>S</i>
LUNA can handle projects containing up to 100 JavaScript files	
UR7-4	<i>C</i>
LUNA can handle projects containing up to 1,000 JavaScript files	
UR7-5	<i>W</i>
LUNA can handle projects containing over 1,000 JavaScript files	
UR7-6	<i>M</i>
LUNA can handle files containing up to 100 LOC	
UR7-7	<i>M</i>
LUNA can handle files containing up to 1,000 LOC	
UR7-8	<i>S</i>
LUNA can handle files containing up to 10,000 LOC	
UR7-9	<i>C</i>
LUNA can handle files containing up to 100,000 LOC	
UR7-10	<i>W</i>
LUNA can handle files containing over 100,000 LOC	

Appendix B

Abstract Syntax Tree

This is the AST from the code snippet of figure 3.1, generated by espreet [24].

```
1 {
2   "type": "Program",
3   "start": 0,
4   "end": 596,
5   "range": [
6     25,
7     596
8   ],
9   "body": [
10    {
11      "type": "VariableDeclaration",
12      "start": 25,
13      "end": 115,
14      "range": [
15        25,
16        115
17      ],
18      "declarations": [
19        {
20          "type": "VariableDeclarator",
21          "start": 31,
22          "end": 114,
23          "range": [
24            31,
25            114
26          ],
27          "id": {
28            "type": "ObjectPattern",
29            "start": 31,
30            "end": 92,
31            "range": [
32              31,
33              92
34            ],
35            "properties": [
36              {
37                "type": "Property",
38                "start": 33,
39                "end": 48,
40                "range": [
41                  33,
```

```

42         48
43     ],
44     "method": false,
45     "shorthand": true,
46     "computed": false,
47     "key": {
48         "type": "Identifier",
49         "start": 33,
50         "end": 48,
51         "range": [
52             33,
53             48
54         ],
55         "name": "constructString"
56     },
57     "kind": "init",
58     "value": {
59         "type": "Identifier",
60         "start": 33,
61         "end": 48,
62         "range": [
63             33,
64             48
65         ],
66         "name": "constructString"
67     }
68 },
69 {
70     "type": "Property",
71     "start": 50,
72     "end": 74,
73     "range": [
74         50,
75         74
76     ],
77     "method": false,
78     "shorthand": true,
79     "computed": false,
80     "key": {
81         "type": "Identifier",
82         "start": 50,
83         "end": 74,
84         "range": [
85             50,
86             74
87         ],
88         "name": "constructTemplateLiteral"
89     },
90     "kind": "init",
91     "value": {
92         "type": "Identifier",
93         "start": 50,
94         "end": 74,
95         "range": [
96             50,
97             74
98         ],
99         "name": "constructTemplateLiteral"

```

```
100     }
101     },
102     {
103         "type": "Property",
104         "start": 76,
105         "end": 90,
106         "range": [
107             76,
108             90
109         ],
110         "method": false,
111         "shorthand": true,
112         "computed": false,
113         "key": {
114             "type": "Identifier",
115             "start": 76,
116             "end": 90,
117             "range": [
118                 76,
119                 90
120             ],
121             "name": "findReferences"
122         },
123         "kind": "init",
124         "value": {
125             "type": "Identifier",
126             "start": 76,
127             "end": 90,
128             "range": [
129                 76,
130                 90
131             ],
132             "name": "findReferences"
133         }
134     }
135 ]
136 },
137 "init": {
138     "type": "CallExpression",
139     "start": 95,
140     "end": 114,
141     "range": [
142         95,
143         114
144     ],
145     "callee": {
146         "type": "Identifier",
147         "start": 95,
148         "end": 102,
149         "range": [
150             95,
151             102
152         ],
153         "name": "require"
154     },
155     "arguments": [
156     {
157         "type": "Literal",
```



```

158         "start": 103,
159         "end": 113,
160         "range": [
161           103,
162           113
163         ],
164         "value": "./common",
165         "raw": "\"./common\""
166       }
167     ]
168   }
169 }
170 ],
171 "kind": "const"
172 },
173 {
174   "type": "VariableDeclaration",
175   "start": 116,
176   "end": 165,
177   "range": [
178     116,
179     165
180   ],
181   "declarations": [
182     {
183       "type": "VariableDeclarator",
184       "start": 122,
185       "end": 164,
186       "range": [
187         122,
188         164
189       ],
190       "id": {
191         "type": "ObjectPattern",
192         "start": 122,
193         "end": 138,
194         "range": [
195           122,
196           138
197         ],
198         "properties": [
199           {
200             "type": "Property",
201             "start": 124,
202             "end": 136,
203             "range": [
204               124,
205               136
206             ],
207             "method": false,
208             "shorthand": true,
209             "computed": false,
210             "key": {
211               "type": "Identifier",
212               "start": 124,
213               "end": 136,
214               "range": [
215                 124,

```

```
216         136
217     ],
218     "name": "extractCalls"
219 },
220 "kind": "init",
221 "value": {
222     "type": "Identifier",
223     "start": 124,
224     "end": 136,
225     "range": [
226         124,
227         136
228     ],
229     "name": "extractCalls"
230 }
231 }
232 ]
233 },
234 "init": {
235     "type": "CallExpression",
236     "start": 141,
237     "end": 164,
238     "range": [
239         141,
240         164
241     ],
242     "callee": {
243         "type": "Identifier",
244         "start": 141,
245         "end": 148,
246         "range": [
247             141,
248             148
249         ],
250         "name": "require"
251     },
252     "arguments": [
253     {
254         "type": "Literal",
255         "start": 149,
256         "end": 163,
257         "range": [
258             149,
259             163
260         ],
261         "value": "./call-graph",
262         "raw": "\\./call-graph\\"
263     }
264     ]
265 }
266 }
267 ],
268 "kind": "const"
269 },
270 {
271     "type": "VariableDeclaration",
272     "start": 166,
273     "end": 215,
```

```

274     "range": [
275         166,
276         215
277     ],
278     "declarations": [
279         {
280             "type": "VariableDeclarator",
281             "start": 172,
282             "end": 214,
283             "range": [
284                 172,
285                 214
286             ],
287             "id": {
288                 "type": "ObjectPattern",
289                 "start": 172,
290                 "end": 187,
291                 "range": [
292                     172,
293                     187
294                 ],
295                 "properties": [
296                     {
297                         "type": "Property",
298                         "start": 174,
299                         "end": 185,
300                         "range": [
301                             174,
302                             185
303                         ],
304                         "method": false,
305                         "shorthand": true,
306                         "computed": false,
307                         "key": {
308                             "type": "Identifier",
309                             "start": 174,
310                             "end": 185,
311                             "range": [
312                                 174,
313                                 185
314                             ],
315                             "name": "extractLibs"
316                         },
317                         "kind": "init",
318                         "value": {
319                             "type": "Identifier",
320                             "start": 174,
321                             "end": 185,
322                             "range": [
323                                 174,
324                                 185
325                             ],
326                             "name": "extractLibs"
327                         }
328                     }
329                 ]
330             },
331             "init": {

```

```
332         "type": "CallExpression",
333         "start": 190,
334         "end": 214,
335         "range": [
336           190,
337           214
338         ],
339         "callee": {
340           "type": "Identifier",
341           "start": 190,
342           "end": 197,
343           "range": [
344             190,
345             197
346           ],
347           "name": "require"
348         },
349         "arguments": [
350           {
351             "type": "Literal",
352             "start": 198,
353             "end": 213,
354             "range": [
355               198,
356               213
357             ],
358             "value": "./library-api",
359             "raw": "\"./library-api\""
360           }
361         ]
362       }
363     }
364   ],
365   "kind": "const"
366 },
367 {
368   "type": "VariableDeclaration",
369   "start": 216,
370   "end": 273,
371   "range": [
372     216,
373     273
374   ],
375   "declarations": [
376     {
377       "type": "VariableDeclarator",
378       "start": 222,
379       "end": 272,
380       "range": [
381         222,
382         272
383       ],
384       "id": {
385         "type": "ObjectPattern",
386         "start": 222,
387         "end": 240,
388         "range": [
389           222,
```

```
390         240
391         ],
392         "properties": [
393         {
394             "type": "Property",
395             "start": 224,
396             "end": 238,
397             "range": [
398                 224,
399                 238
400             ],
401             "method": false,
402             "shorthand": true,
403             "computed": false,
404             "key": {
405                 "type": "Identifier",
406                 "start": 224,
407                 "end": 238,
408                 "range": [
409                     224,
410                     238
411                 ],
412                 "name": "getNodeModules"
413             },
414             "kind": "init",
415             "value": {
416                 "type": "Identifier",
417                 "start": 224,
418                 "end": 238,
419                 "range": [
420                     224,
421                     238
422                 ],
423                 "name": "getNodeModules"
424             }
425         }
426     ]
427 },
428 "init": {
429     "type": "CallExpression",
430     "start": 243,
431     "end": 272,
432     "range": [
433         243,
434         272
435     ],
436     "callee": {
437         "type": "Identifier",
438         "start": 243,
439         "end": 250,
440         "range": [
441             243,
442             250
443         ],
444         "name": "require"
445     },
446     "arguments": [
447     {
```

```
448         "type": "Literal",
449         "start": 251,
450         "end": 271,
451         "range": [
452             251,
453             271
454         ],
455         "value": "./dependency-graph",
456         "raw": "\\./dependency-graph\\"
457     }
458 ]
459 }
460 }
461 ],
462 "kind": "const"
463 },
464 {
465     "type": "VariableDeclaration",
466     "start": 312,
467     "end": 386,
468     "range": [
469         312,
470         386
471     ],
472     "declarations": [
473         {
474             "type": "VariableDeclarator",
475             "start": 318,
476             "end": 385,
477             "range": [
478                 318,
479                 385
480             ],
481             "id": {
482                 "type": "ObjectPattern",
483                 "start": 318,
484                 "end": 367,
485                 "range": [
486                     318,
487                     367
488                 ],
489                 "properties": [
490                     {
491                         "type": "Property",
492                         "start": 320,
493                         "end": 328,
494                         "range": [
495                             320,
496                             328
497                         ],
498                         "method": false,
499                         "shorthand": true,
500                         "computed": false,
501                         "key": {
502                             "type": "Identifier",
503                             "start": 320,
504                             "end": 328,
505                             "range": [
```

```

506         320,
507         328
508     ],
509     "name": "basename"
510 },
511 "kind": "init",
512 "value": {
513     "type": "Identifier",
514     "start": 320,
515     "end": 328,
516     "range": [
517         320,
518         328
519     ],
520     "name": "basename"
521 }
522 },
523 {
524     "type": "Property",
525     "start": 330,
526     "end": 337,
527     "range": [
528     330,
529     337
530     ],
531     "method": false,
532     "shorthand": true,
533     "computed": false,
534     "key": {
535     "type": "Identifier",
536     "start": 330,
537     "end": 337,
538     "range": [
539     330,
540     337
541     ],
542     "name": "dirname"
543     },
544     "kind": "init",
545     "value": {
546     "type": "Identifier",
547     "start": 330,
548     "end": 337,
549     "range": [
550     330,
551     337
552     ],
553     "name": "dirname"
554     }
555 },
556 {
557     "type": "Property",
558     "start": 339,
559     "end": 346,
560     "range": [
561     339,
562     346
563     ],

```

```

564         "method": false,
565         "shorthand": true,
566         "computed": false,
567         "key": {
568         "type": "Identifier",
569         "start": 339,
570         "end": 346,
571         "range": [
572             339,
573             346
574         ],
575         "name": "extname"
576     },
577     "kind": "init",
578     "value": {
579     "type": "Identifier",
580     "start": 339,
581     "end": 346,
582     "range": [
583         339,
584         346
585     ],
586     "name": "extname"
587     }
588 },
589 {
590     "type": "Property",
591     "start": 348,
592     "end": 356,
593     "range": [
594         348,
595         356
596     ],
597     "method": false,
598     "shorthand": true,
599     "computed": false,
600     "key": {
601     "type": "Identifier",
602     "start": 348,
603     "end": 356,
604     "range": [
605         348,
606         356
607     ],
608     "name": "relative"
609     },
610     "kind": "init",
611     "value": {
612     "type": "Identifier",
613     "start": 348,
614     "end": 356,
615     "range": [
616         348,
617         356
618     ],
619     "name": "relative"
620     }
621 },

```



```

622     {
623         "type": "Property",
624         "start": 358,
625         "end": 365,
626         "range": [
627             358,
628             365
629         ],
630         "method": false,
631         "shorthand": true,
632         "computed": false,
633         "key": {
634             "type": "Identifier",
635             "start": 358,
636             "end": 365,
637             "range": [
638                 358,
639                 365
640             ],
641             "name": "resolve"
642         },
643         "kind": "init",
644         "value": {
645             "type": "Identifier",
646             "start": 358,
647             "end": 365,
648             "range": [
649                 358,
650                 365
651             ],
652             "name": "resolve"
653         }
654     }
655 ]
656 },
657 "init": {
658     "type": "CallExpression",
659     "start": 370,
660     "end": 385,
661     "range": [
662         370,
663         385
664     ],
665     "callee": {
666         "type": "Identifier",
667         "start": 370,
668         "end": 377,
669         "range": [
670             370,
671             377
672         ],
673         "name": "require"
674     },
675     "arguments": [
676     {
677         "type": "Literal",
678         "start": 378,
679         "end": 384,

```

```
680         "range": [
681             378,
682             384
683         ],
684         "value": "path",
685         "raw": "\"path\""
686     }
687 ]
688 }
689 }
690 ],
691 "kind": "const"
692 },
693 {
694     "type": "VariableDeclaration",
695     "start": 387,
696     "end": 432,
697     "range": [
698         387,
699         432
700     ],
701     "declarations": [
702         {
703             "type": "VariableDeclarator",
704             "start": 393,
705             "end": 431,
706             "range": [
707                 393,
708                 431
709             ],
710             "id": {
711                 "type": "ObjectPattern",
712                 "start": 393,
713                 "end": 412,
714                 "range": [
715                     393,
716                     412
717                 ],
718                 "properties": [
719                     {
720                         "type": "Property",
721                         "start": 395,
722                         "end": 410,
723                         "range": [
724                             395,
725                             410
726                         ],
727                         "method": false,
728                         "shorthand": true,
729                         "computed": false,
730                         "key": {
731                             "type": "Identifier",
732                             "start": 395,
733                             "end": 410,
734                             "range": [
735                                 395,
736                                 410
737                             ],
```

```

738         "name": "generateFlatAST"
739     },
740     "kind": "init",
741     "value": {
742         "type": "Identifier",
743         "start": 395,
744         "end": 410,
745         "range": [
746             395,
747             410
748         ],
749         "name": "generateFlatAST"
750     }
751 }
752 ]
753 },
754 "init": {
755     "type": "CallExpression",
756     "start": 415,
757     "end": 431,
758     "range": [
759         415,
760         431
761     ],
762     "callee": {
763         "type": "Identifier",
764         "start": 415,
765         "end": 422,
766         "range": [
767             415,
768             422
769         ],
770         "name": "require"
771     },
772     "arguments": [
773     {
774         "type": "Literal",
775         "start": 423,
776         "end": 430,
777         "range": [
778             423,
779             430
780         ],
781         "value": "flast",
782         "raw": "\\\"flast\\\""
783     }
784 ]
785 }
786 }
787 ],
788 "kind": "const"
789 },
790 {
791     "type": "VariableDeclaration",
792     "start": 433,
793     "end": 471,
794     "range": [
795         433,

```

```
796         471
797     ],
798     "declarations": [
799         {
800             "type": "VariableDeclarator",
801             "start": 439,
802             "end": 470,
803             "range": [
804                 439,
805                 470
806             ],
807             "id": {
808                 "type": "ObjectPattern",
809                 "start": 439,
810                 "end": 452,
811                 "range": [
812                     439,
813                     452
814                 ],
815                 "properties": [
816                     {
817                         "type": "Property",
818                         "start": 441,
819                         "end": 450,
820                         "range": [
821                             441,
822                             450
823                         ],
824                         "method": false,
825                         "shorthand": true,
826                         "computed": false,
827                         "key": {
828                             "type": "Identifier",
829                             "start": 441,
830                             "end": 450,
831                             "range": [
832                                 441,
833                                 450
834                             ],
835                             "name": "promisify"
836                         },
837                         "kind": "init",
838                         "value": {
839                             "type": "Identifier",
840                             "start": 441,
841                             "end": 450,
842                             "range": [
843                                 441,
844                                 450
845                             ],
846                             "name": "promisify"
847                         }
848                     }
849                 ]
850             },
851             "init": {
852                 "type": "CallExpression",
853                 "start": 455,
```

```

854         "end": 470,
855         "range": [
856           455,
857           470
858         ],
859         "callee": {
860           "type": "Identifier",
861           "start": 455,
862           "end": 462,
863           "range": [
864             455,
865             462
866           ],
867           "name": "require"
868         },
869         "arguments": [
870           {
871             "type": "Literal",
872             "start": 463,
873             "end": 469,
874             "range": [
875               463,
876               469
877             ],
878             "value": "util",
879             "raw": "\"util\""
880           }
881         ]
882       }
883     ],
884     "kind": "const"
885   },
886   {
887     "type": "VariableDeclaration",
888     "start": 472,
889     "end": 522,
890     "range": [
891       472,
892       522
893     ],
894     "declarations": [
895       {
896         "type": "VariableDeclarator",
897         "start": 478,
898         "end": 521,
899         "range": [
900           478,
901           521
902         ],
903         "id": {
904           "type": "ObjectPattern",
905           "start": 478,
906           "end": 496,
907           "range": [
908             478,
909             496
910           ],
911

```

```
912         "properties": [  
913     {  
914         "type": "Property",  
915         "start": 480,  
916         "end": 488,  
917         "range": [  
918             480,  
919             488  
920         ],  
921         "method": false,  
922         "shorthand": true,  
923         "computed": false,  
924         "key": {  
925             "type": "Identifier",  
926             "start": 480,  
927             "end": 488,  
928             "range": [  
929                 480,  
930                 488  
931             ],  
932             "name": "readFile"  
933         },  
934         "kind": "init",  
935         "value": {  
936             "type": "Identifier",  
937             "start": 480,  
938             "end": 488,  
939             "range": [  
940                 480,  
941                 488  
942             ],  
943             "name": "readFile"  
944         }  
945     },  
946     {  
947         "type": "Property",  
948         "start": 490,  
949         "end": 494,  
950         "range": [  
951             490,  
952             494  
953         ],  
954         "method": false,  
955         "shorthand": true,  
956         "computed": false,  
957         "key": {  
958             "type": "Identifier",  
959             "start": 490,  
960             "end": 494,  
961             "range": [  
962                 490,  
963                 494  
964             ],  
965             "name": "stat"  
966         },  
967         "kind": "init",  
968         "value": {  
969             "type": "Identifier",
```

```

970         "start": 490,
971         "end": 494,
972         "range": [
973             490,
974             494
975         ],
976         "name": "stat"
977     }
978 }
979 ]
980 },
981 "init": {
982     "type": "CallExpression",
983     "start": 499,
984     "end": 521,
985     "range": [
986         499,
987         521
988     ],
989     "callee": {
990         "type": "Identifier",
991         "start": 499,
992         "end": 506,
993         "range": [
994             499,
995             506
996         ],
997         "name": "require"
998     },
999     "arguments": [
1000     {
1001         "type": "Literal",
1002         "start": 507,
1003         "end": 520,
1004         "range": [
1005             507,
1006             520
1007         ],
1008         "value": "fs/promises",
1009         "raw": "\"fs/promises\""
1010     }
1011     ]
1012 }
1013 }
1014 ],
1015 "kind": "const"
1016 },
1017 {
1018     "type": "VariableDeclaration",
1019     "start": 523,
1020     "end": 552,
1021     "range": [
1022         523,
1023         552
1024     ],
1025     "declarations": [
1026     {
1027         "type": "VariableDeclarator",

```

```
1028     "start": 529,
1029     "end": 551,
1030     "range": [
1031         529,
1032         551
1033     ],
1034     "id": {
1035         "type": "Identifier",
1036         "start": 529,
1037         "end": 533,
1038         "range": [
1039             529,
1040             533
1041         ],
1042         "name": "glob"
1043     },
1044     "init": {
1045         "type": "CallExpression",
1046         "start": 536,
1047         "end": 551,
1048         "range": [
1049             536,
1050             551
1051         ],
1052         "callee": {
1053             "type": "Identifier",
1054             "start": 536,
1055             "end": 543,
1056             "range": [
1057                 536,
1058                 543
1059             ],
1060             "name": "require"
1061         },
1062         "arguments": [
1063             {
1064                 "type": "Literal",
1065                 "start": 544,
1066                 "end": 550,
1067                 "range": [
1068                     544,
1069                     550
1070                 ],
1071                 "value": "glob",
1072                 "raw": "\"glob\""
1073             }
1074         ]
1075     }
1076 }
1077 ],
1078 "kind": "const"
1079 },
1080 {
1081     "type": "VariableDeclaration",
1082     "start": 553,
1083     "end": 596,
1084     "range": [
1085         553,
```



```

1086         596
1087     ],
1088     "declarations": [
1089         {
1090             "type": "VariableDeclarator",
1091             "start": 559,
1092             "end": 595,
1093             "range": [
1094                 559,
1095                 595
1096             ],
1097             "id": {
1098                 "type": "Identifier",
1099                 "start": 559,
1100                 "end": 570,
1101                 "range": [
1102                     559,
1103                     570
1104                 ],
1105                 "name": "randomColor"
1106             },
1107             "init": {
1108                 "type": "CallExpression",
1109                 "start": 573,
1110                 "end": 595,
1111                 "range": [
1112                     573,
1113                     595
1114                 ],
1115                 "callee": {
1116                     "type": "Identifier",
1117                     "start": 573,
1118                     "end": 580,
1119                     "range": [
1120                         573,
1121                         580
1122                     ],
1123                     "name": "require"
1124                 },
1125                 "arguments": [
1126                     {
1127                         "type": "Literal",
1128                         "start": 581,
1129                         "end": 594,
1130                         "range": [
1131                             581,
1132                             594
1133                         ],
1134                         "value": "randomcolor",
1135                         "raw": "\"randomcolor\""
1136                     }
1137                 ]
1138             }
1139         }
1140     ],
1141     "kind": "const"
1142 }
1143 ],

```

```
1144 |   "sourceType": "module"  
1145 | }
```

Listing B.1: AST of the snippet of the LUNA's scanner.js

Appendix C

Form

This is a Google Form to gather feedback for LUNA. It is available online¹.

C.1 Questions

¹<https://forms.gle/8JS12P2uxU8nWBRx7>

LUNA (Library Usage in Node.js Analyzer)

This will roughly take 15-30 minutes to complete. Thank you so much for your time!

* Required

Demographic (1/5)

Please tell us about you

1. What is your occupation? *

2. What is your programming experience? *

Mark only one oval.

1 2 3 4 5

Novice Expert

3. What is your experience in JavaScript? *

Mark only one oval.

1 2 3 4 5

Novice Expert

4. Do you know what the following concepts are?

Select all that apply

Check all that apply.

- GitHub
- Software library
- ECMAScript
- Node.js
- NPM
- package.json
- RequireJS

LUNA (2/5)

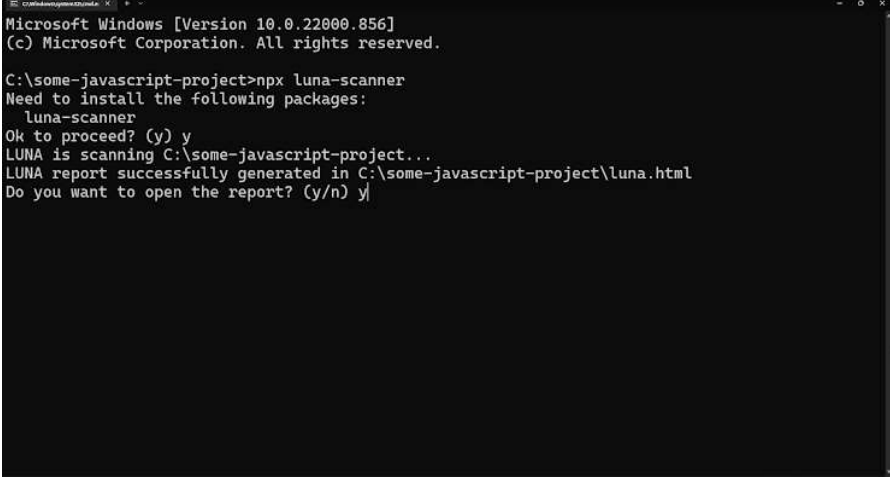
Library Usage in Node.js Analyzer (LUNA) is a software development tool for node.js projects, with a focus on libraries. The goal of LUNA is to aid developers in better understanding how libraries are being utilized in their projects.

HOW TO: GENERATE REPORT

1. Install [node.js](#) (ideally 16.17.0 LTS or higher) for your operating system
2. Have a node.js project ready to scan (tip: it can be LUNA itself)
3. Open a terminal (cmd on Windows)
4. Execute:
`cd <path_to_your_project>`
(change <path_to_your_project> to the full path of your node.js project)
5. Execute:
`npx luna-scanner`
6. If prompted, accept installing dependencies (enter: y)
7. Wait until LUNA report is generated
8. Examine your generated LUNA report (see below)

HOW TO: USE LUNA REPORT

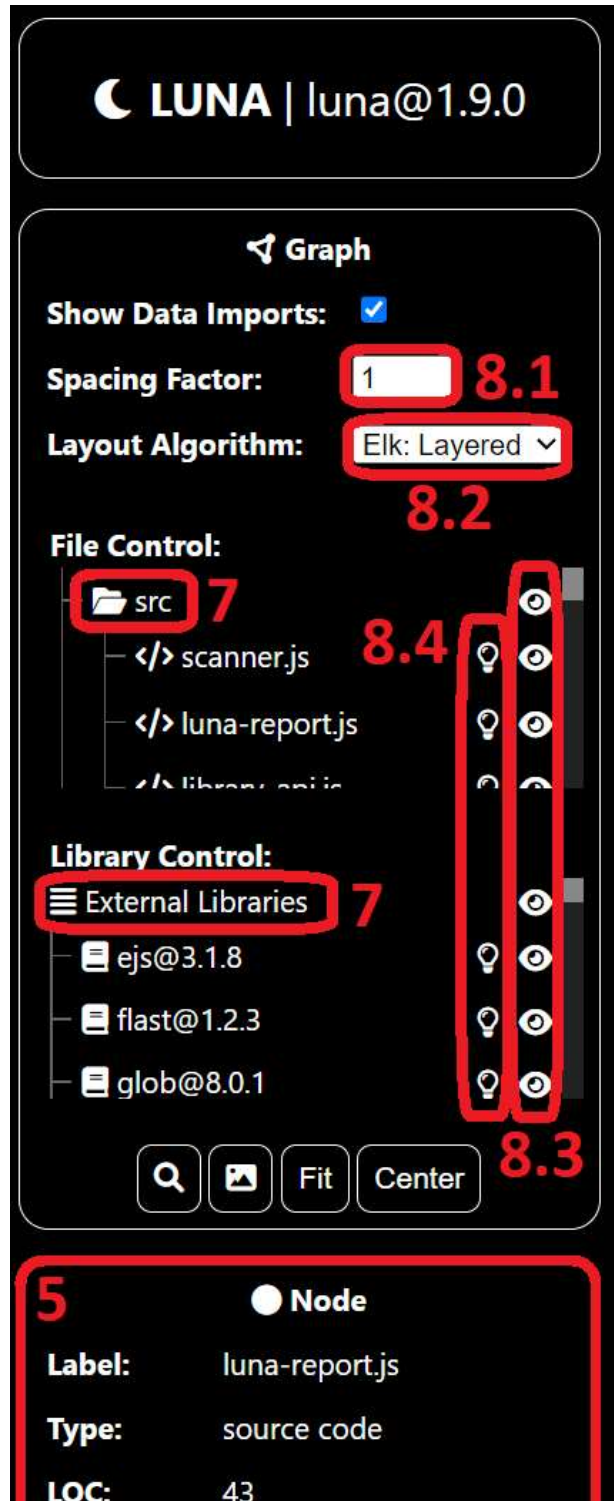
1. Open the LUNA report (using a chromium based web browser)
2. Wait for LUNA to finish loading and positioning nodes in the graph
3. Drag (click and hold) the mouse to pan around or move nodes around
4. Use the mouse wheel to zoom in or out
5. Hovering over a node will display information on the bottom left and highlight connected nodes
6. Using Shift + Click on a node will lock it, so that focus remains on this node (Shift + Click node again to unlock)
7. Double Click nodes or groups in the graph or menu to collapse/expand them
8. Use the menu on the left to manipulate the graph:
 - 8.1. Adjust the scale of the graph / space between nodes
 - 8.2. Adjust the layout of the graph / position of the nodes
 - 8.3. Hide a selection of nodes (representing libraries or files)
 - 8.4. Highlight a selection of nodes (representing libraries or files)
9. Hover your mouse above menu items to find more information about their functionality

Terminal to run LUNA

```
Microsoft Windows [Version 10.0.22000.856]
(c) Microsoft Corporation. All rights reserved.

C:\some-javascript-project>npx luna-scanner
Need to install the following packages:
  luna-scanner
Ok to proceed? (y) y
LUNA is scanning C:\some-javascript-project...
LUNA report successfully generated in C:\some-javascript-project\luna.html
Do you want to open the report? (y/n) y
```

LUNA report menu (numbers refer to *HOW TO: USE LUNA REPORT*)





User
Tasks
(3/5)

To get yourself familiar with LUNA's functionality, we designed a few tasks. Please attempt to perform the following tasks and answer the questions below. For these tasks specifically, you will need to scan LUNA itself. You can download the LUNA project from here:

<https://github.com/royvandijk06/luna> ([↓ zip](#))

Please refer to the previous page for the instructions.

Task 1

Use LUNA to find what library or libraries used by LUNA are affected by a security vulnerability in the dependency 'estraverse@5.2.0'.

Task 2

Use LUNA to determine what files and functions inside LUNA's source code need to be changed when removing or replacing the library 'flast'.

Task 3

Use LUNA to find which 5 API features of the internal library 'path' is being used by LUNA's code.

Task 4

Use LUNA to figure out the topological order in which the 7 js files inside the 'src' folder from LUNA use each other using a suitable layout, starting with `src/index.js`.

Task 5

You want to show a project collaborator a nice image of only the inner workings of `src/call-graph.js` from LUNA's source code. Use LUNA to do this.

5. Did you manage to succeed in each task with LUNA? *

Mark only one oval per row.

	Success	Fail	Unsure
Task 1	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Task 2	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Task 3	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Task 4	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Task 5	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

6. How difficult was each task to do with LUNA? *

Mark only one oval per row.

	Easy	Normal	Hard
Task 1	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Task 2	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Task 3	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Task 4	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Task 5	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

7. **Task 1**

If you struggled with this task, what part was difficult to do?

Also, what improvement(s) could be made to LUNA to make this task easier, if any?

8. **Task 2**

If you struggled with this task, what part was difficult to do?

Also, what improvement(s) could be made to LUNA to make this task easier, if any?

9. **Task 3**

If you struggled with this task, what part was difficult to do?

Also, what improvement(s) could be made to LUNA to make this task easier, if any?

10. **Task 4**

If you struggled with this task, what part was difficult to do?

Also, what improvement(s) could be made to LUNA to make this task easier, if any?

11. **Task 5**

If you struggled with this task, what part was difficult to do?

Also, what improvement(s) could be made to LUNA to make this task easier, if any?

**System
Usability (4/5)**

After using LUNA, please select how much you agree or disagree with the following statements.

12. I think that I would like to use LUNA frequently. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

13. I found LUNA unnecessarily complex. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

14. I thought LUNA was easy to use. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

15. I think that I would need the support of a technical person to be able to use LUNA. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

16. I found the various functions in LUNA were well integrated. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

17. I thought there was too much inconsistency in LUNA. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

18. I would imagine that most people would learn to use LUNA very quickly. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

19. I found LUNA very cumbersome to use. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

20. I felt very confident using LUNA. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

21. I needed to learn a lot of things before I could get going with LUNA. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

System Feedback (5/5)	After using LUNA, please let us know what do you think about it.
-----------------------	--

22. Would you use LUNA in the future? If so, for what purpose?

23. What do you like most about LUNA?

24. What do you dislike most about LUNA / What can be improved?

Please consider each of the following features of LUNA:

1. Layout
2. nodes
3. Edges/connections
4. Info/stats
5. Control
6. UI
7. Graph design

25. What do you think about the analytical features of LUNA, i.e. all the information that it provides?

Like, is there any information that you think is missing?

26. What do you think about the visualization of LUNA, i.e. how the information is presented?

27. Do you have any other feedback for LUNA?
Anything left unsaid?

This content is neither created nor endorsed by Google.

Google Forms

C.2 Answers

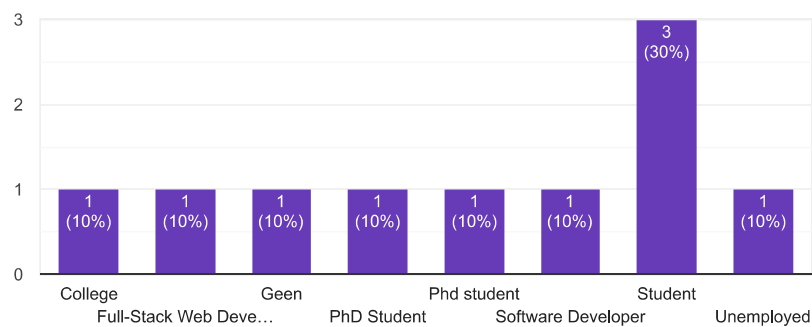
LUNA (Library Usage in Node.js Analyzer)

10 responses

Demographic (1/5)

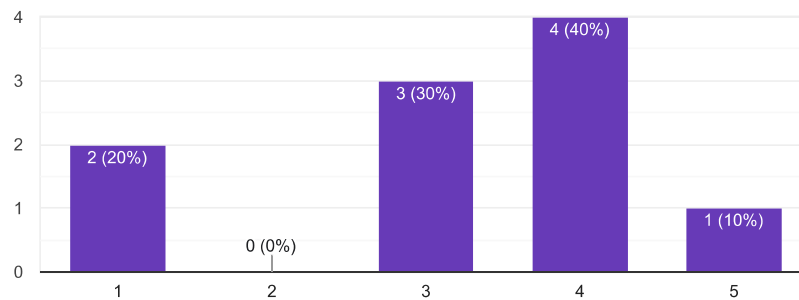
What is your occupation?

10 responses



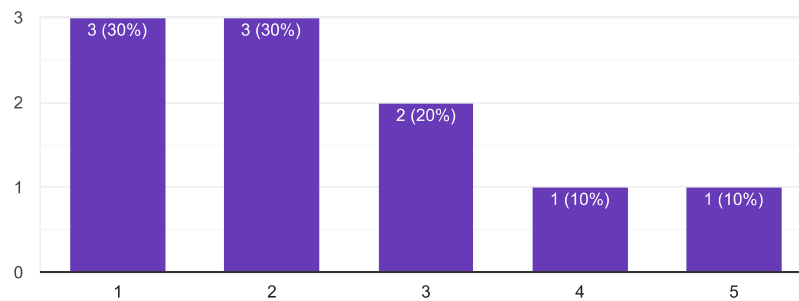
What is your programming experience?

10 responses



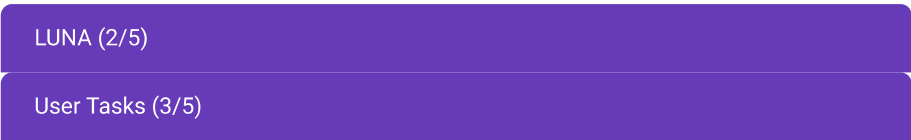
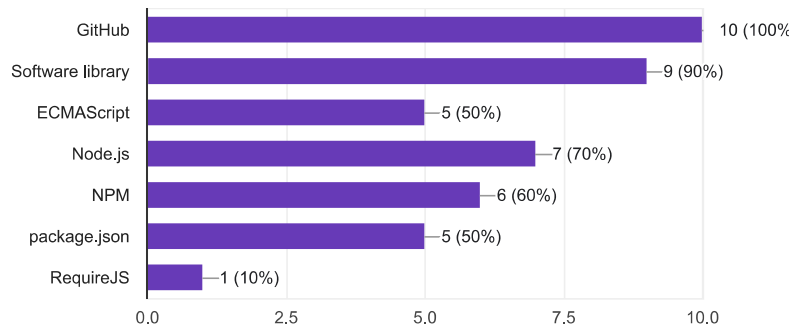
What is your experience in JavaScript?

10 responses

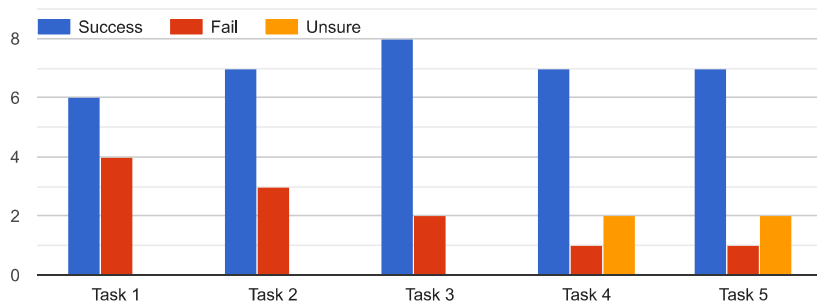


Do you know what the following concepts are?

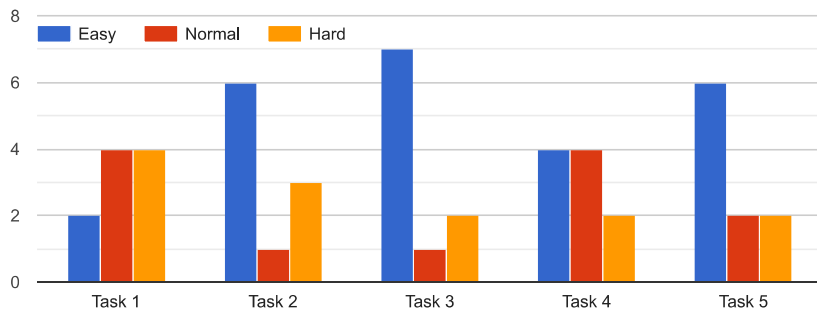
10 responses



Did you manage to succeed in each task with LUNA?



How difficult was each task to do with LUNA?



Task 1

If you struggled with this task, what part was difficult to do?

Also, what improvement(s) could be made to LUNA to make this task easier, if any?

6 responses

It could be useful to visually filter on a search term, in order to avoid having a large amount of irrelevant libraries on the screen.

It took a little while to figure out that I needed to double-click the dependency tree node to expand it. Maybe the search button on the left bar could always search inside of collapsed nodes and notify you if the search results include items within collapsed nodes.

I could not get the software installed or the project opened in the first place, I kept getting errors.

Search doesn't find such library, can't find it in the graph

Expanding the dependency tree and locking the node (it was described in the document but it is not completely intuitive). The search option could also be improved. Felt that I was only searching on the left-hand side tree.

Knowledge insufficient

Task 2

If you struggled with this task, what part was difficult to do?

Also, what improvement(s) could be made to LUNA to make this task easier, if any?

6 responses

I wasn't able to figure out how to make the report show specifically which file(s) import flast. Expanding the flast node shows that generateFlatAST is the API used when flast is imported, but I can't see which file is calling that. All other nodes go dark when hovering over flast.

prima te doen, maar een bug(?) met uit en in klappen maakte het wat lastiger

I could not get the software installed or the project opened in the first place, I kept getting errors.

Double-clicking on the node to see the function level was quite intuitive. Maybe you can filter the functions that are not relevant for the use case scenario.

Some UI element to indicate that a item is expandable could be helpful

Knowledge insufficient

Task 3

If you struggled with this task, what part was difficult to do?

Also, what improvement(s) could be made to LUNA to make this task easier, if any?

4 responses

I could not get the software installed or the project opened in the first place, I kept getting errors.

No info on how to find it

What do you mean by an API feature? I would keep the same visualization as we had in the previous task when interacting with the source code. I am missing the invocations and other API uses performed in the source code.

Knowledge insufficient

Task 4

If you struggled with this task, what part was difficult to do?

Also, what improvement(s) could be made to LUNA to make this task easier, if any?

4 responses

Layout still had a few of overlaps, it would have been better if those are avoided

I could not get the software installed or the project opened in the first place, I kept getting errors.

Collapsing interaction was a bit difficult (don't know where to double click). The use of the layout was quite straight forward, though.

Knowledge insufficient to know if succesful

Task 5

If you struggled with this task, what part was difficult to do?

Also, what improvement(s) could be made to LUNA to make this task easier, if any?

6 responses

When hiding a folder, toggle the visibility of all files inside, this makes showing a single file easier

Rerun layout algorithm is huidige geen optie, je moet om het algoritme opnieuw een layout te laten genereren wisselen naar een andere layout en dan terug (4 handelingen ipv 1)

I could not get the software installed or the project opened in the first place, I kept getting errors.

It was smooth.

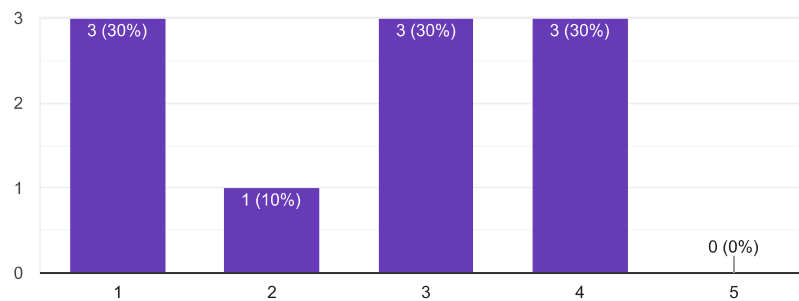
Option to hide many files at once

Knowledge insufficient to know if succesful

System Usability (4/5)

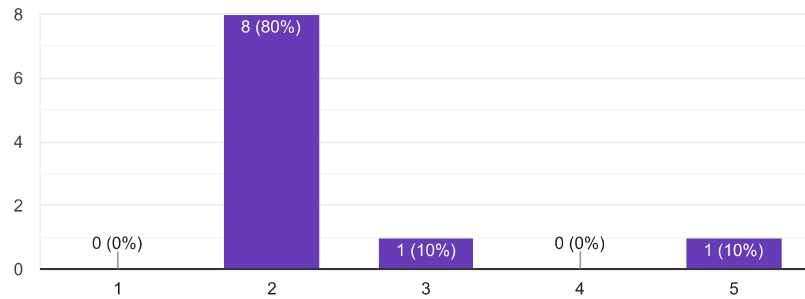
I think that I would like to use LUNA frequently.

10 responses



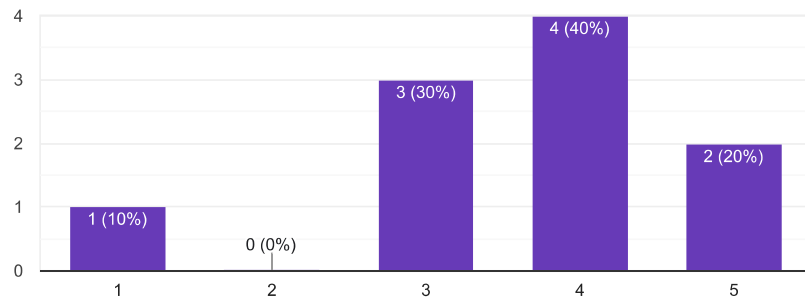
I found LUNA unnecessarily complex.

10 responses



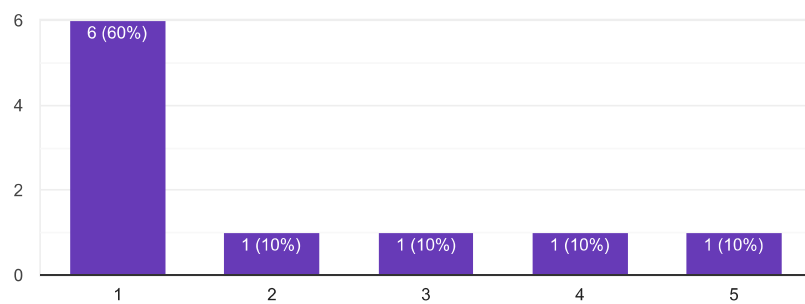
I thought LUNA was easy to use.

10 responses



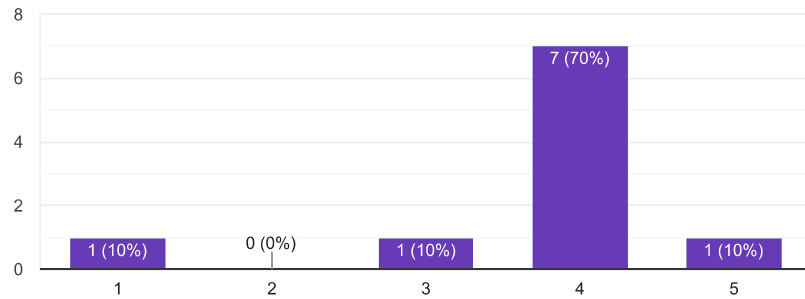
I think that I would need the support of a technical person to be able to use LUNA.

10 responses



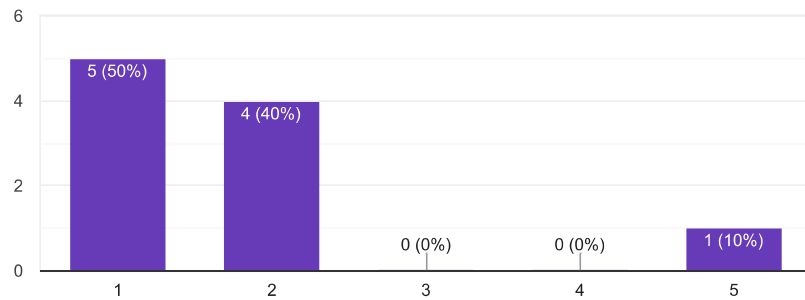
I found the various functions in LUNA were well integrated.

10 responses



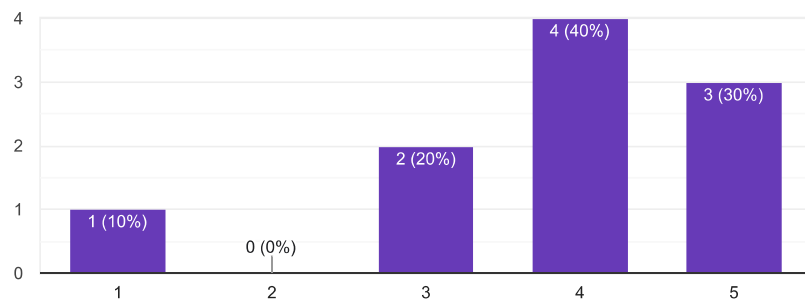
I thought there was too much inconsistency in LUNA.

10 responses



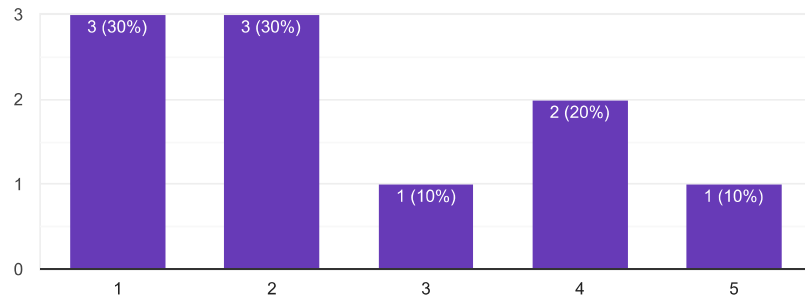
I would imagine that most people would learn to use LUNA very quickly.

10 responses



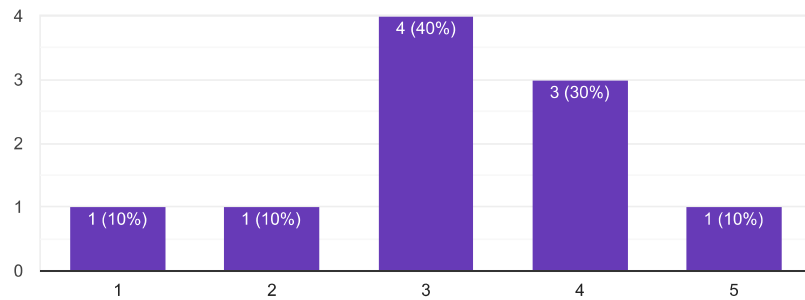
I found LUNA very cumbersome to use.

10 responses



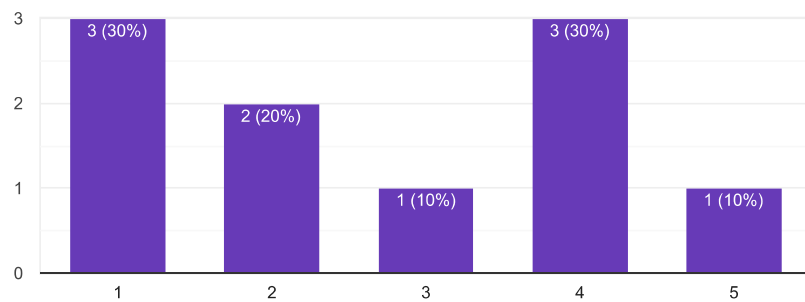
I felt very confident using LUNA.

10 responses



I needed to learn a lot of things before I could get going with LUNA.

10 responses



System Feedback (5/5)

Would you use LUNA in the future? If so, for what purpose?

10 responses

no, no reason to

Probably not

I think I would. I maintain a number of increasingly-complex Node.js library modules and getting a visual indication of how the inner workings interact can be useful.

Ja, duidelijk maken en plannen van architectuur.
en als je code aanpast kun je de dependencies vrij makkelijk vinden

No, I now understand that my programming knowledge is severely lacking.

Probably not due to not using much JS

To see how different modules are correlated, and seeing which parts of my code use a certain library/method when refactoring

Yes, If i need to verify how the upgrade of library impacts my own code.

I would if I used JS. Probably mostly to keep track of how my own code is interconnected.

I would after acquiring further knowledge on programming. The visualization aspect works very well. Other functions i need to learn more before using.

What do you like most about LUNA?

9 responses

It mostly does one thing: visualize the dependencies in a Node.js project, and it does it well

Being able to visualize interdependencies in a graph can be valuable for figuring out how different components interact.

Gemak van instantane graphs

I cannot say, I could not get it installed.

It's simplicity

As above - ability to see how different modules are correlated, and seeing which parts of my code use a certain library/method when refactoring

It is quite interactive and it is easy to spot the causes of certain issues coming from library-client interaction.

It gives a good visual insight into your code

The visualization of the different parts of a program.

What do you dislike most about LUNA / What can be improved?

9 responses

When expanding files or folders, the boxes sometimes overlap or get positioned in what seems to me a strange layout

The UI is a little clunky at times. I'd like to be able to right-click-drag to move the entire graph without moving any nodes relative to other nodes. For example, if I'm zoomed in to a node such that the background is no longer visible, I might want to be able to pan the entire view without moving the node I'm zoomed into.

It could also be helpful to be able to disable visualizing links between specific nodes. For example, I might want to be able to see links between source files in /src without also having the links to dependencies cluttering the screen.

geen rmb opties, het voelt als een natuurlijke manier om alle opties met een element te verkennen en vinden

The fact that I was confronted with my limited comprehension of programming after all these years of not doing it anymore.

Cola takes ages to scale fully leading to annoyance attempting to zoom in

Right now it doesn't seem to support the following:

- Non-pure js projects - React, Vue, typescript?
- Yarn workspaces / monorepos

A limited number of actions might not be that intuitive (like locking a node). The interaction is good but can be still optimized. The GUI is well designed in my opinion.

The layout could be improved to make things more visually structured. Also more visual indication of the functionality through UI elements

7

What do you think about the analytical features of LUNA, i.e. all the information that it provides?

9 responses

Dependency analysis can be useful, depending on the project. LUNA seems to collect the relevant data to do this.

It's potentially useful. If I'm refactoring a function and need to see everywhere it's called, I'm more likely to just use my IDE's find usages feature, but having a visual graph can be helpful too.

dat ze waarschijnlijk werken, maar dat ik niet precies weet wat ik er mee zou doen

-

No

It has enough data to be very useful

I think it provides very important information about internal and external dependencies. It is quite useful to see the overall picture of a project. Information about possible threats (breaking changes, security vulnerabilities, etc.).

I think it is very useful, I can't think of anything that is missing.

Insufficient knowledge to find anything missing.

What do you think about the visualization of LUNA, i.e. how the information is presented?

9 responses

Apart from the flaws mentioned earlier, I think the visualization is sufficient for its purpose.

It's well-presented, and having different layouts available is good. Even if I don't specific what every layout means from its name, just clicking through them until I find a layout that fits my present needs is helpful.

Goed, maar pijlen uitzetten zou kunnen helpen, en de hoeveelheid pijlen kan voor lag. als het programma in zijn animatie zit om dingen te herordenen kun je ook niet veel doen (niet perse veel aan te doen, maar soms onhandig)

-

It's well presented

No issues

I like it pretty much. Well distributed and clean. Would give the option for a lighter theme.

There are many different icons, it is a bit visually cluttered. But overall the information visually presented is useful

All different aspects are presented in a nice uncluttered manner.

Do you have any other feedback for LUNA?

5 responses

bugs maken dingen soms onbetrouwbaar en unresponsive, herladen van de pagina of CTRL+R drukken gaat gelukkig wel snel

-

It's a very nice polished app

Not currently

Great job! Like it a lot.

This content is neither created nor endorsed by Google. [Report Abuse](#) - [Terms of Service](#) - [Privacy Policy](#).

Google Forms

