



University of Western Macedonia  
Department of Informatics & Telecommunications Engineering

Kozani, 2017

# Software Evolution

Diploma Thesis

CHATZIMPARMPAS ANGELOS

SRN: 638

Supervisor

---

Bibi Stamatia

**Title:** Software Evolution

**Description:** Diploma thesis within the framework of studies for the title of the diploma «Informatics and Telecommunications Engineer.»

**Keywords:** Software Evolution, JavaScript, Lehman's Laws, GitHub, Open Source Software, SonarQube, JSClassFinder

**Author:** Chatzimpampas Angelos

**Date of creation:** 29-08-2017

**Year of issue:** 2017

**Country of issue:** GR

**Text language:** Eng

The present diploma thesis was elaborated within the framework of the studies for the diploma awarded by the University of Western Macedonia entitled «Informatics and Telecommunications Engineer.»

Approved on .././2017 by a committee of inquiry consisting of:

Full name:

Academic rank:

Signature:

- 1.
- 2.
- 3.

## **Acknowledgement**

I would first like to thank my thesis supervisor lecturer Mrs. Bibi Stamatia of the Department of Informatics & Telecommunications Engineering at the University of Western Macedonia. The door to Prof. Bibi's office was always open whenever I ran into a trouble spot or had a question about my research or writing. She consistently allowed this diploma thesis to be my work but steered me in the right direction whenever he thought I needed it.

I would also like to thank all my friends for the real support and encouragement they provided to me. Moreover, the teamwork and beautiful moments I passed with them in various projects and at the same time out of the Department of Informatics & Telecommunications Engineering is without a doubt one of the most unforgettable moments of my life.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Chatzimparmpas Angelos

Kozani, August 2017

## Abstract

The primary goal of the present Diploma Thesis is the development of a web application that predicts trends in software development in open source programs. This platform was designed to serve and meet the needs of developers or researchers who want to control the applications they create and how they eventually evolve these over time with new releases. The platform has been designed to work locally on a computer or online if it is placed on a server that has the power to control the software that developers build. The conclusions that one of the above users may make is vital for the development of the software they produce or for further research on new trends in the development of the software of various projects developers.

More and more software is written in programming languages such as JavaScript. Open source programs are continually evolving, making it difficult to anticipate success and progress, which is ultimately influenced by a variety of sectors and factors. It is therefore crucial and necessary to develop a platform that will control, analyze and study these changes successfully and of course providing benefits for developers but at the same time supplying data with future research.

Summarizing the essential elements of this Diploma Thesis, we focus on four main topics. Precisely, in the first topic, investigation, and analysis of similar issues related to various open source software and how they evolve from their creation years to the present. Most of them are written in programming languages, which are fundamental and have been created many years ago. Also, the purpose and goal of creating this research and the systems that make up the search for results are clear. The second topic presents the requirements and the programs used as well as data for their complete installation. The third topic shows the platform and its use with precise examples and explanations for each system that has been created. Finally, the fourth topic analyzes and gives the results of the research that was built on the subject of "Software Evolution."

**Keywords:** Software Evolution, JavaScript, Lehman's Laws, GitHub, Open Source Software, SonarQube, JSClazzFinder

## Εξέλιξη Έργων Λογισμικού

### Περίληψη

Η παρούσα διπλωματική εργασία έχει ως βασικό στόχο την ανάπτυξη μιας διαδικτυακής εφαρμογής η οποία προβλέπει τις τάσεις στην ανάπτυξη λογισμικού σε προγράμματα ανοικτού κώδικα. Η πλατφόρμα αυτή σχεδιάστηκε έτσι ώστε να εξυπηρετεί και να ικανοποιεί τις ανάγκες των προγραμματιστών ή ερευνητών που θέλουν να ελέγχουν τα προγράμματα που δημιουργούν και το πόσο τελικά εξελίσσονται με την πάροδο του χρόνου και των νέων εκδόσεων. Η πλατφόρμα έχει δημιουργηθεί για να λειτουργεί τοπικά σε έναν υπολογιστή ή διαδικτυακά αν τοποθετηθεί σε έναν διακομιστή ο οποίος θα έχει την ισχύ ώστε να ελέγχει το λογισμικό που κατασκευάζουν προγραμματιστές. Τα συμπεράσματα τα οποία μπορεί να βγάλει ένας από τους παραπάνω χρήστες είναι σημαντικά για την εξέλιξη του λογισμικού που παράγουν ή και για περαιτέρω έρευνα πάνω σε νέες τάσεις για την ανάπτυξη του λογισμικού έργων διαφόρων παραγωγών.

Ολοένα και περισσότερα λογισμικά γράφονται σε γλώσσες προγραμματισμού όπως η JavaScript. Τα προγράμματα ανοικτού κώδικα συνεχώς εξελίσσονται κάτι που κάνει δύσκολο να προβλεφθεί η επιτυχία και η πορεία αυτών η οποία τελικά επηρεάζεται από ποικίλους τομείς και παράγοντες. Συνεπώς είναι καίριο και απαραίτητο να δημιουργηθεί μια πλατφόρμα η οποία θα ελέγχει, θα αναλύει και θα μελετάει τις αλλαγές αυτές με επιτυχία και φυσικά παρέχοντας οφέλη για τους προγραμματιστές αλλά ταυτόχρονα να τροφοδοτεί με στοιχεία μελλοντικές έρευνες.

Συνοψίζοντας τα βασικά στοιχεία αυτής της διπλωματικής εργασίας, επικεντρώνονται σε τέσσερις θεματικούς άξονες. Συγκεκριμένα, στον πρώτο άξονα, γίνεται διερεύνηση και ανάλυση παρόμοιων θεμάτων που αφορούν διάφορα λογισμικά ανοικτού κώδικα και το πως εξελίσσονται από τα χρόνια δημιουργίας τους μέχρι και σήμερα. Τα περισσότερα από αυτά είναι γραμμένα σε γλώσσες προγραμματισμού οι οποίες είναι βασικές και έχουν δημιουργηθεί εδώ και πολλά χρόνια. Επιπλέον, γίνεται ξεκάθαρος ο λόγος και ο στόχος δημιουργίας της έρευνας αυτής και των συστημάτων που την απαρτίζουν για την εύρεση αποτελεσμάτων. Στο δεύτερο άξονα γίνεται η παρουσίαση των απαιτήσεων και των προγραμμάτων που χρησιμοποιούνται καθώς και στοιχεία για την πλήρη εγκατάσταση αυτών. Στον τρίτο άξονα παρουσιάζεται αναλυτικά η πλατφόρμα και η χρήση της με ακριβή παραδείγματα και επεξηγήσεις για το κάθε σύστημα που έχει δημιουργηθεί. Τέλος, στον τέταρτο άξονα γίνεται ανάλυση και παρουσίαση των αποτελεσμάτων της έρευνας που δημιουργήθηκε με θέμα την «Εξέλιξη Έργων Λογισμικού».

**Λέξεις κλειδιά:** Εξέλιξη Έργων Λογισμικού, JavaScript, Νόμοι του Lehman, GitHub, Λογισμικό Ανοικτού Κώδικα, SonarQube, JSClassFinder

## Contents

<b>1. INTRODUCTION .....</b>	<b>12</b>
<b>1.1 Objective.....</b>	<b>13</b>
<b>1.2 Organization of Chapters .....</b>	<b>13</b>
<b>2. THEORETICAL BACKGROUND .....</b>	<b>14</b>
<b>2.1 Lehman's Laws of Software Evolution &amp; Observations for Contemporary Applications .....</b>	<b>14</b>
<b>2.1.1 Lehman's Laws of Software Evolution .....</b>	<b>14</b>
<b>2.1.2 Observations for Contemporary Applications.....</b>	<b>15</b>
<b>2.2 Software Evolution in Other Programming Languages and Comparison with JavaScript .....</b>	<b>17</b>
<b>2.2.1 Software Evolution in Other Programming Languages .....</b>	<b>17</b>
<b>2.2.1.1 Evolution of applications written in C.....</b>	<b>18</b>
<b>2.2.1.2 Evolution of applications written in PHP .....</b>	<b>19</b>
<b>2.2.2 PHP in comparison with JavaScript as languages for open-source projects .....</b>	<b>20</b>
<b>2.3 Details about JavaScript Language and GitHub Software Development Platform .....</b>	<b>22</b>
<b>2.3.1 Details about JavaScript Language .....</b>	<b>22</b>
<b>2.3.2 Key Features of JavaScript .....</b>	<b>22</b>
<b>2.3.3 GitHub Software Development Platform.....</b>	<b>24</b>
<b>3. ANALYSIS AND DESIGN OF THE WEB APPLICATION .....</b>	<b>25</b>
<b>3.1 Description of the Requirements.....</b>	<b>25</b>
<b>3.2 Technology and Tools .....</b>	<b>25</b>
<b>3.2.1 Operating Systems .....</b>	<b>25</b>
<b>3.2.2 XAMPP – The Most Popular PHP Development Environment.....</b>	<b>25</b>
<b>3.2.3 LARAVEL and MySQL.....</b>	<b>26</b>
<b>3.2.4 Goutte - A Simple PHP Web Scrapper.....</b>	<b>26</b>
<b>3.2.5 Git – Version Control System .....</b>	<b>26</b>
<b>3.2.6 SonarQube Continuous Code Quality .....</b>	<b>26</b>
<b>3.2.7 JSClassFinder – A Tool to Detect Class-Like Structures in JavaScript.....</b>	<b>28</b>
<b>3.2.8 AutoHotkey – The Ultimate Automation Scripting Language for Windows .....</b>	<b>29</b>
<b>3.2.9 Grafana – The Open Platform for Beautiful Analytics and Monitoring .....</b>	<b>29</b>
<b>3.3 The JS evolution Tool .....</b>	<b>30</b>
<b>4. PLATFORM PRESENTATION .....</b>	<b>36</b>

4.1 Creation of the Project and Basic Metrics with the Automated Procedure .....	36
4.2 Generating Basic Metrics with the Manual Procedure .....	38
4.3 Calculations from the Existing Measurements .....	41
4.4 Receiving the Contributors and Stats of Them for Every Project .....	42
4.5 JSClassFinder Variables Added to the Thesis Database .....	42
4.6 General Fixes to Common Problems .....	48
4.7 Creation of Charts with the Use of Grafana.....	49
4.8 Important Parts of Code.....	50
4.9 Details of the Values (Metrics) Obtained .....	52
5. RESULTS AND CONCLUSIONS .....	58
5.1 General Stats for JS projects.....	58
5.2 Database Results Preparation for Analysis.....	62
5.3 Validation of the Software Evolution.....	66
5.3.1 Law I: Continuing Change .....	66
5.3.2 Law II: Increasing complexity .....	68
5.3.3 Law III: Self-Regulation .....	70
5.3.4 Law IV: Conservation of Organizational Stability .....	72
5.3.5 Law V: Conservation of Familiarity.....	76
5.3.6 Law VI: Continuing Growth .....	77
5.3.7 Law VII: Declining Quality .....	79
5.3.8 Law VIII: Feedback System.....	83
5.4 Conclusions and Comparison to other studies.....	84
Bibliographic References .....	86



Figure 1 : JSClassFinder’s architecture [42] ..... 28

Figure 2: JS tool combined usage ..... 31

Figure 3: Starting XAMPP and SonarQube..... 32

Figure 4: The entire thesis folder ..... 33

Figure 5: Git clone of a repository ..... 34

Figure 6: SonarQube Scanner configurations ..... 34

Figure 7: Four main folders necessary for the analysis ..... 35

Figure 8: AutoHotkey and two necessary scripts ..... 35

Figure 9: Projects table in Database ..... 36

Figure 10: Main thesis functionality through creation of a new Project..... 37

Figure 11: For every release stats and metrics in Database ..... 37

Figure 12: Zeros in a line in Database at 1285 identifier and version name 2.9.1.1 ..... 38

Figure 13: Specific’s release tag from GitHub official website ..... 39

Figure 14: Setting release’s tag manually to Git for SonarQube Scanner ..... 39

Figure 15: Manual execution of SonarQube Scanner ..... 40

Figure 16: Save results from SonarQube server to HTML format ..... 41

Figure 17: Store results in the Database for a specific identifier ..... 41

Figure 18: Perform calculations for all the releases ..... 41

Figure 19: Store the maximum of 100 contributors in the Database..... 42

Figure 20: Generate a file from every JavaScript file in a release ..... 42

Figure 21: Parse the generic file and export the AST tree ..... 43

Figure 22: Copy and paste the AST tree to a JSON file ..... 44

Figure 23: Send JSON file to JSClassFinder ..... 44

Figure 24: Execution of JSClassFinder image and Pharo..... 45

Figure 25: Run ast file for analysis ..... 46

Figure 26: Results from JSClassFinder software ..... 47

Figure 27: Search for “function” key-word in GitHub for every release ..... 47

Figure 28: Store the results in the Database ..... 48

Figure 29: Fixing an underlying problem of automated procedure ..... 49

Figure 30: Connection to Database and Grafana ..... 50

Figure 31: Creation of charts via SQL queries..... 50

Figure 32: Paths of basic code segments ..... 51

Figure 33: Part of code that might change for some GitHub projects ..... 52

Figure 34: Download of zip file for every release ..... 52

Figure 35: Number of releases for each project ..... 59

Figure 36: Number of commits for each project ..... 59

Figure 37: Number of watches for each project..... 60

Figure 38: Number of forks created for each project..... 60

Figure 39: Number of Stars for each project ..... 61

Figure 40: Number of open issues for each project .....	61
Figure 41: Number of closed issues for each project .....	62
Figure 42: Selection of table .....	62
Figure 43: Configurations of the export .....	63
Figure 44: The last configuration for the export to be performed.....	63
Figure 45: Delimited cells.....	64
Figure 46: Semicolon option for separation of the entire first column.....	65
Figure 47: DBR chart for 40 Projects.....	68
Figure 48: Complexity chart for 40 Projects .....	70
Figure 49: Incremental Growth chart for 40 Projects.....	72
Figure 50: Maintenance Effort chart for 40 Projects .....	75
Figure 51: Number of Commits chart for 40 Projects.....	76
Figure 52: Incremental Changes chart for 40 Projects .....	77
Figure 53: Lines of Code chart for 40 Projects.....	79

Table 1: Studies on the validity of Lehman’s laws .....	16
Table 2: The laws of software evolution of the FW analysis [61] .....	17
Table 3: Summary of statistical hypothesis for each Lehman law and each application [12].....	18
Table 4: Data Analysis [13].....	20
Table 5: Top 15 most popular languages used on GitHub in the last twelve months of the Octoverse 2016* .....	21
Table 6: JavaScript and PHP fundamental features differences.....	24
Table 7: SonarQube Metrics .....	27
Table 8: Presentation of results .....	29
Table 9 ( <b>Releases Stats Table</b> ): Metrics used to this thesis for validation of each Lehman law to every GitHub JavaScript program.....	56
Table 10 ( <b>Projects Table</b> ): General statistics metrics for every Project that has been tested .....	56
Table 11 ( <b>Contributors Table</b> ): Top 100 contributors to a Project with most commits.....	57
Table 12: Statistical results on law I (continuing change). .....	68
Table 13: Statistical results on law II (increasing complexity). .....	70
Table 14: Statistical results on law III (self-regulation).....	72
Table 15: Statistical results on law IV (conservation of organizational stability). .....	75
Table 16: Statistical results on law V (conservation of familiarity). .....	77
Table 17: Statistical results on law VI (continuing growth). .....	79
Table 18: Statistical results on law VII (declining quality). .....	83
Table 19: Statistical results on law VIII (feedback system). .....	84
Table 20: Validation of the laws .....	85
Table 21: Studies about the validity of Lehman’s laws including ours .....	85

## 1. INTRODUCTION

Software evolution is the term applied in software engineering and to be more specific in software maintenance branch and is related to the method of developing software initially, then frequently updating it for many reasons. As Fred Brooks declares in his book, over 90% of the costs in a software system is given in the maintenance stage and software will unavoidable be maintained [62]. The maintenance of software is divided into four main categories [63]:

- Corrective maintenance: Identified problems are solved after of course being known through reactive adjustment of a software product.
- Adaptive maintenance: To hold a software product usable in a modified or modifying environment after of course being known, adjustment of a software product must be performed.
- Perfective maintenance: To enhance performance or maintainability after of course being known, adjustment of a software product must be done.
- Preventive maintenance: Adjustment of a software product must be performed to search and find potential threats in the software product before they grow to sufficient errors.

There are small additions to each category, but the scientists kept four basic types that are presented above. To evaluate these categories in real-life software, a set of behaviors in the evolution of proprietary software has been identified by Lehman and his colleagues.

Several researchers are focused on the subject of “Software Evolution” and tried to test the validity of Lehman’s laws. However, JavaScript is a well-known and widely used programming language almost no research has been conducted for JS Software Evolution. This diploma thesis achieved to test JavaScript open-source projects from GitHub and draw conclusions about their current state. Also, with these results, the future of JavaScript software can be depicted.

More and more software is written in JavaScript programming language. Therefore, it is important to test how well made and faultlessly are some of the best known of them and how they evolve. It is, thus, necessary to develop a platform that will test this software in such a way as to achieve this goal successfully, while at the same time promotes further research on such an important issue.

## 1.1 Objective

The primary goal of the present diploma thesis is to develop an online web application that, in combination with other tools, will provide complete data on “Software Evolution” of JavaScript programs. Specifically, in its context, the theoretical background of “Software Evolution” is analyzed with lots of details. Furthermore, the results of the survey are interpreted and presented with high tech and popular tools. Also, the description, design, and implementation of the platform created to test the open source software are presented.

This platform was designed firstly to cover my and Mrs. Bibi’s curiosity referring to “Software” and how it will evolve in the future. In an extension of that, it also serves and meets the needs of software developers and researchers. The platform that we managed to create will help researchers to estimate and understand main problems of open source projects that programmers developed. This thesis provides a complete package of programs like the web application that is created with a particular purpose to collect data from JavaScript notable projects through the use of special software. Through specific URLs, an entirely open source project gets into testing, and the results of the procedure are stored in the database.

## 1.2 Organization of Chapters

This diploma thesis is divided into five chapters-thematic sections.

In 1<sup>st</sup> chapter, we have a brief reference to “Software Evolution.” In addition, there is a short presentation of the subject of the thesis and the reasons for choosing the exact web application.

In 2<sup>nd</sup> chapter, a more general analysis is made of the term “Software Evolution.” Furthermore, relevant work is examined, and the theoretical background that a reader might need is provided.

In 3<sup>rd</sup> chapter, the requirements of the platform are presented. Then, theoretical details about systems that the web application uses are shown. At the end of the section, an extensive guide of how to set up and execute all the software that this diploma thesis provides is given to the reader.

In the 4<sup>th</sup> chapter, the platform is presented in detail as well as all the features it has. With the help of screenshots, all the platform's capabilities are captured, and a detailed explanation of how to use it is provided.

In the 5<sup>th</sup> chapter, a detailed exhibition of the results and conclusions that emerged from the use of the web application is presented.

Finally, reference is made to all the bibliographic sources used in the dissertation, as well as to the websites that helped to develop the web application.

## 2. THEORETICAL BACKGROUND

This specific chapter defines the meanings and the necessary theoretical background to understand the diploma thesis fully. Firstly, everything starts from the education and the factors that led Lehman to study the software evolution. Furthermore, the focus of this survey is to test the validity of the Lehman's Laws which are eight in number. Last but not least is the explanation of the reasons that the above tests have been applied to JavaScript projects in comparison with other surveys and more details about JavaScript as a language.

### 2.1 Lehman's Laws of Software Evolution & Observations for Contemporary Applications

#### 2.1.1 Lehman's Laws of Software Evolution

Lehman and Belady [1], the mid-70s, formulated the laws of software evolution [1] that is the most well-known work for them. A software system that everyone thinks it's going to work cannot be for sure guaranteed that it will. Two reasons are supporting the previous claim:

- The first is that the real world is very complicated and no one could know what will happen to inside a running program.
- The second reason is that the software interferes with the environment and the surroundings in which it will run [1].

When time passes the program gets feedback from its users. This continuous feedback supports developers to evolve the program and the source code eventually. Even if an application accomplishes users requirements, it should continue to develop because changes to the environment will appear. Lehman understood the difficulty of improving a program, and with that in mind, he created some laws to summarize his conclusions for E-type software systems [1]. E-type systems are those embedded and actively worked in a real-world area [51]. The "E" letter on E-type systems stands for "Evolutionary." E-type programs are being influenced by the surroundings and need to be adaptive. Furthermore, Evolutionary-type software is the most common and important referring to the real and research world respectively [1]. To continue with, these results are well-known as Lehman's Laws of Software Evolution (adapted from [1, 2, 3]) and are summarized as follows:

- 1) "Continuing Change" - A program should adapt to the new needs of the users or else it will be progressively less satisfactory for them.
- 2) "Increasing Complexity" - A program will have its complexity constantly increasing except precise work is done to maintain or even decrease it.

- 3) "Self-Regulation" - The software systems are self-regulating while evolving with close to the normal process of measures and distribution of the product.
- 4) "Conservation of Organizational Stability" - The average effective global activity rate doesn't change when time pass for an E-type evolving system. That means the work stays the same for every release.
- 5) "Conservation of Familiarity" - An E-type software system has the same or even less new content over time in comparison with every successive release.
- 6) "Continuing Growth" - Over its lifetime a program must show a progressive increase in the number of functional content to maintain its users pleased.
- 7) "Declining Quality" - New operational limitations will appear, and an E-type system should be maintained and adapted to these but in general, the quality will decline over time, and nothing will happen to preserve it.
- 8) "Feedback System" - An E-type program constitutes of multi-agent, multi-loop, multi-level feedback systems and must be treated like this to achieve extravagant improvements over time. This law proves the importance of the user to give feedback to the evolutionary system to improve and update in the future.

**2.1.2 Observations for Contemporary Applications**

Lehman laws, presented in the previous section, are aspired by the software development techniques adopted in IBM where he was working. The data he used to evaluate these laws were derived from big industrial programs developed with old-fashioned methods for system-oriented software. However, the software has suffered significant changes, now it is more agile, cloud-based and has the power to run in multiple environments and is highly professional and technical. With all that in mind, it is necessary to test and validate each one of the Lehman's Laws merely to understand which should be adjusted for the new software and its evolution [1].

Some studies doubt the truth of some laws, for example, the 3<sup>rd</sup>, 4<sup>th</sup>, 5<sup>th</sup>. The results of these studies can be shown in the following table [13]:

Reference	Year	Programming Language	Number of Projects	I	II	III	IV	V	VI	VII	VIII
Godfrey & Tu [52, 53]	2000 and 2001	C	5	Y			N		Y		N
Robles et al. [54]	2005	C, C++, Java	19	Y			N		Y		N

Mens et al. [55]	2008	Java	1	Y	N				Y		
Xie et al. [56]	2009	C	7	Y	Y	Y		N	Y	N	N
Israeli & Feiteison [57]	2010	C	1	Y	N	Y	Y		Y	N	Y
Businge et al. [58]	2010	Java	21	Y		Y		N	Y		
Neamtiu et al. [59]	2013	C	9	Y	N	N	N	N	Y	N	N
Kaur et al. [60]	2014	C++	2	Y	Y	Y		Y	Y	Y	
Amanatidis & Chatzigeorgiou 2015[13]	2015	PHP	30	Y	N	Y	Y	Y	Y		N

*Table 1: Studies on the validity of Lehman's laws*

These studies found that some rules are valid for example those who have the “Y” letter. The “N” letter is for the invalid laws. Lastly, the empty rows aren't tested, or the results are doubtful about the validity of each law.

That mainly happens due to the different working styles of the past where developers and engineers were working as teams to significant and expensive projects with deadlines. Of course, the above conclusions could be understood if we compare the past with the new industrial software development that is being used in modern times. These new industrial software development also copy the open source systems style which has more freedom to it. So, when a program manager sees a significant improvement, releases the new version and doesn't need to have a deadline [1].

Lehman did some studies and used some metrics to calculate and test the Laws. One Lehman's study has the following results from various measurements in a system which is called Logica FW (Fastwire) [61] for every release:



No.	Brief Name	Support	Indicator
I	Continuing Change	√	Fig. 3 clearly indicates continuing growth. Logica's confirmation that this is partly due to adaptation and change supports the law. Quantification will be of interest.
II	Increasing Complexity	√	The inverse square law of growth (eq. 1) and its predictive power (fig. 7) supports complexity as a constraining factor.
III	Self Regulation	?	The ripple (fig. 3) of the, otherwise, smooth growth (eq. 1) suggests regulation around a smooth trend. Identification of the underlying mechanisms is required to support the law as it stands.
IV	Conservation of Organisational Stability (invariant work rate)	√	The ability to obtain a close fit and very good predictive power with a single and constant parameter $E$ (eq. 1) provides support. Measures of the work rate are required.
V	Conservation of Familiarity	?	Fig. 5 still suggests that the average incremental growth has a definite trend. Its invariance as in the original formulation is now, however, questioned. Determination of the trend and the consequences of a release whose incremental growth exceeds the average significantly must await the further behaviour of the system in its evolution.
VI	Continuing Growth	√	Fig. 3 clearly indicates continuing growth. Logica's confirmation that this is partly due to functional growth supports the law. Quantification will be of interest.
VII	Declining Quality	?	No data that provides evidence for or against is available.
VIII	Feedback System	√	Regulation as in figs. 3, 5, 7, 8 and inverse square law, (eq. 1) are supportive. Feedback control mechanisms must be identified to obtain further support.

Table 2: The laws of software evolution of the FW analysis [61]

Most of the laws were validated back in 1997 for the “logica” FW system. Laws 3, 5, 7 are in doubt. But the most important thing is that this survey was in a specific software system which makes the results controversial. These metrics Lehman used for the previous conclusions are being used identically to new studies, and that’s why the discussion of some concerns is required, but for now, the concentration goes to the efficacy of the Lehman's Laws.

## 2.2 Software Evolution in Other Programming Languages and Comparison with JavaScript

### 2.2.1 Software Evolution in Other Programming Languages

There are many surveys about Lehman's Laws to open-source programs, but most of them are in C and PHP languages. JavaScript is a language that hasn't been tested abroad yet. The previous assertion is the main reason that this thesis tries to identify the validity and get results of the laws for a prevalent but still not examined programming language. On the following chapters, there are results from two significant studies.

### 2.2.1.1 Evolution of applications written in C

The 1<sup>st</sup> paper is called “Towards a better understanding of software evolution: an empirical study on open-source software” and is about nine open-source applications written in C. The authors checked if some hypotheses are right for every law and conclude to the confirmation or doubt of the laws individually [12]. The first column presents laws and their names. The second column is the hypothesis of each law with a number which is the indicator of the exact law (i.e., H1) and letters to count the metrics for each law (i.e., H2a, H2b). Each hypothesis has a description just to know what it is for. The rest columns are the nine projects that they tested for validation. The "Y" means that for an application it is confirmed and the "N" means that it is rejected. The results can be shown in the following table [12]:

Law	Hypothesis, metric	Bash	BIND	Bison	OpenSSH	Quagga	Samba	Sendmail	SQLite	Vsftpd
I <i>Continuing change</i>	$H^1$ : cumulative changes	Y	Y	Y	Y	Y	Y	Y	Y	Y
II <i>Increasing complexity</i>	$H^{2a}$ : calls per function	N	Y	N	N	N	N	N	N	Y
	$H^{2b}$ : cyc. complexity (absolute)	Y	Y	Y	Y	Y	Y	Y	Y	Y
	$H^{2c}$ : cyc. complexity (normalized)	N	N	Y	Y	N	N	N	Y	N
	$H^{2d}$ : common coupling (absolute)	Y	Y	Y	Y	Y	Y	Y	Y	Y
III <i>Self regulation</i>	$H^{2e}$ : common coupling (normalized)	N	N	N	N	Y	N	N	N	N
	$H^{2f}$ : number of modules	Y	Y	Y	Y	Y	Y	Y	Y	N
IV <i>Conservation of org. stability</i>	$H^{2g}$ : number of functions	Y	Y	Y	Y	N	Y	Y	Y	Y
	$H^{2h}$ : changes per day	N	N	N	N	N	N	N	N	N
V <i>Conservation of familiarity</i>	$H^{2i}$ : change rate	N	N	N	N	N	N	N	N	N
	$H^{2j}$ : growth rate	N	N	N	N	N	N	N	N	N
VI <i>Continuing growth</i>	$H^{2k}$ : net module growth	N	N	N	N	N	N	N	N	N
	$H^{2l}$ : growth rate (new functions)	N	N	N	N	N	N	N	N	N
VII <i>Declining quality</i>	$H^{2m}$ : number of changes	N	N	N	N	N	N	N	N	N
	$H^{2n}$ : LOC	Y	Y	Y	Y	Y	Y	Y	Y	Y
VIII <i>Feedback system</i>	$H^{2o}$ : number of modules	Y	Y	Y	Y	Y	Y	Y	Y	Y
	$H^{2p}$ : number of definitions	Y	Y	Y	Y	Y	Y	Y	Y	Y
VIII <i>Feedback system</i>	$H^{2q}$ : number of defects	N	N	N	N	N	N	N	N	N
	$H^{2r}$ : defect density (by LOC)	N	N	N	N	N	N	N	N	N
	$H^{2s}$ : defect density (by $\Delta$ LOC)	N	N	N	N	N	N	N	N	N
	$H^{2t}$ : internal quality, see Law II	N	N	N	N	N	N	N	N	N
VIII <i>Feedback system</i>	$H^{2u}$ : number of modules $\propto \sqrt[3]{RSN}$	Y	Y	Y	Y	Y	Y	Y	Y	Y
	$H^{2v}$ : $\frac{\Delta S}{S} \propto t^{-2/3}$ (number of modules)	N	N	N	N	Y	N	N	N	Y
	$H^{2w}$ : $\frac{\Delta S}{S} \propto t^{-2/3}$ (LOC)	N	N	N	N	N	N	N	N	Y
VIII <i>Feedback system</i>	$H^{2x}$ : $\frac{\Delta S}{S} \propto t^{-2/3}$ (number of functions)	N	Y	N	N	Y	N	N	N	Y

Table 3: Summary of statistical hypothesis for each Lehman law and each application [12]

The methodology they used was to first get the data from the official versions of each of the nine open-source programs. Then they processed and merged all the code to a single .c file, but at the same time, they held module information. Afterwards, the procedure was to run source code analysis tools which are two in number [12]:

- ASTdiff gathers a variety of change metrics, for example, changes in attributes, methods, types, etc. With this tool, they collected information about code complexity and modules.
- RSM stands for Resource Standard Metrics and is a commercial tool that they used for cyclomatic computing complexity [12].

The last step was to use statistical hypothesis testing to validate and draw conclusions about the nine projects. There were four kinds of analysis depending on the type of each hypothesis [12]:

- Increase/decrease test: A univariate linear regression has been performed to test the changes of a metric. The dependent variable is the metric they tested. On the contrary, the independent argument is the number of days since the start of the project for that release or maybe the release's sequence number. If  $b > 0$  or  $b < 0$  ( $b$  is the slope) they had to increase and decrease respectively and  $p\text{-value} < 0.05$  to get the hypothesis validated.
- Non-zero test: One sample t-test has been performed to test if a metric value has non-zero values. The specified value was zero, and the null hypothesis is that each release has a mean equal to zero. If this assumption was wrong ( $p\text{-value} < 0.05$ ) the specific release has non-zero values.
- Invariance test: Some laws support that a specific metric is invariant over time. To test this hypothesis they used Levene's test for equality of variance in two samples. The first example contains the exact metric values for all releases and the second has everything the same but no variation which means that all elements are equal to the first example. If the two samples have similar deviation and  $p\text{-value} < 0.05$  the hypothesis is validated.
- Non-linear growth test: They performed a univariate linear regression where the argument is the value of the metric (e.g., LOC) for a specific release, and the independent argument is the growth model (e.g., the square root of time). The hypothesis is validated if  $p\text{-value} < 0.05$ .

P-value is an exact threshold that they set at 0.05 and has been used to every hypothesis.

To make it more "clear" the 1<sup>st</sup> and 6<sup>th</sup> laws are only confirmed by this paper [12]. Hypothesis 1, 2d, 3a, 3b, 6a, 6b, 6c and 8a are confirmed from the analysis they performed in each project. In controversy, 2a, 2c, 4a, 4b, 4c, 5a, 5b, 5c, 7a, 7b, 7c, 8b and 8c hypothesis are not confirmed. The remaining testing hypothesis is partially being confirmed in a portion of projects. To have a fully confirmed law each of the hypothesis for this law must be validated. For example, 6a, 6b, 6c are validated, and that's why the 6<sup>th</sup> law is being confirmed.

### 2.2.1.2 Evolution of applications written in PHP

The 2<sup>nd</sup> paper is called "Studying the evolution of PHP web applications" and is about thirty open-source applications written in PHP. They checked some variables (metrics) for every law and found results for the confirmation of the laws [13]. The conclusions of this paper can be shown in the following table [13]:

## Software Evolution

Laws	Variables	Data analysis
Law I (Continuing change)	[V <sub>1</sub> ] Days Between Releases (DBR)	- Trend test - Slope estimation
Law II (Increasing complexity)	[V <sub>2</sub> ] Complexity metric: Cyclomatic Complexity Number/Lines Of Code (CCN/LOC)	- Trend test - Slope estimation
Law III (Self regulation)	[V <sub>3</sub> ] Incremental growth of methods & functions	- Trend test - Slope estimation
Law IV (Conservation of organizational stability)	[V <sub>4.1</sub> ] Maintenance effort: Effort = total changes/DBR[V <sub>4.2</sub> ] Number of commits	- Trend test - Slope estimation
Law V (Conservation of familiarity)	[V <sub>5</sub> ] Incremental changes (IC) in methods & functions	- Trend test - Slope estimation
Law VI (Continuing growth)	[V <sub>6</sub> ] Lines of Code (LOC)	- Trend test - Slope estimation
Law VII (Declining quality)	[V <sub>7.1</sub> ] Afferent Coupling (CA)* [V <sub>7.2</sub> ] Efferent Coupling (CE)* [V <sub>7.3</sub> ] Depth of Inheritance Tree (DIT)* [V <sub>7.4</sub> ] Comment Ratio (CR); Commented Lines Of Code/Lines Of Code [V <sub>7.5</sub> ] Maintainability Index (MI) [V <sub>7.6</sub> ] Number of bug-related commits	- Trend test - Slope estimation
Law VIII (Feedback system)	[V <sub>8</sub> ] Actual ( $\frac{c}{t}$ ) and theoretical growth rate ( $c \cdot t^{-\frac{1}{2}}$ )	two sample Kolmogorov-Smirnoff test

\* These metrics have been measured at class level and their average values (divided by the number of classes) have been considered.

*Table 4: Data Analysis [13]*

The findings were that 1st, 3rd, 4th, 5th and 6th laws are confirmed, and the others aren't [13].

### 2.2.2 PHP in comparison with JavaScript as languages for open-source projects

JavaScript is a universal language for web front-end applications but has also embedded the Node.js in 2009 which is for server-side scripting [24]. PHP and JS have some similarities which are (adapted from [25, 26]):

- Both languages are usually used for the web and were developed specifically for it in 1995.
- The syntax styles are taken from the C language on both of them.
- Until lately PHP wasn't Object-Oriented language, and both weren't formally Object-Oriented languages.
- They are platform independent, but PHP needs a compiler and JS a run-time environment.

There are also many differences between them and here are some basic:

- PHP use is mainly for server-side things while JavaScript is for client-side but with the Node.js JavaScript has also been a server-side scripting language.
- JavaScript only has constructors and functions in contrast to PHP which has and uses classes.
- JavaScript is used for visual effects and improvements of web GUIs.
- Users can deactivate all JavaScript while browsing the internet because it is a client-side language and has some features that are being analyzed in the 2.4.2 chapter.

Table 5 shows us that JavaScript is the number one language compared to the others and various GitHub applications are written in it.

Programming Language	Pull Requests	Percentage Changes from Previous Period
JavaScript	1,604,219	+97%
Java	763,783	+63%
Python	744,045	+54%
Ruby	740,610	+66%
PHP	478,153	+43%
C++	330,259	+43%
CSS	271,782	+36%
C#	229,985	+88%
C	202,295	+47%
Go	188,121	+93%
Shell	143,071	+76%
Objective C	75,378	+37%
Scala	70,216	+54%
Swift	62,284	+262%
TypeScript	55,587	+250%

*Table 5: Top 15 most popular languages used on GitHub in the last twelve months of the Octoverse 2016\**

\*Stats for the Table 5 are obtained from “The state of the Octoverse 2016” (<https://octoverse.github.com/>).

The JS has advantages for the programmers and users which can be summed up in the following bullet points:

- An easy language because it is effortless to learn and the syntax is approaching English.
- Instant response for every visitor because without server interactions you don’t have to wait for pages to reload to get what you are requesting.
- Pretty fast for the end-user because scripts are being executed on the user’s computer and for a portion of results they are immediately presented.
- Interactivity is raised because of the development of interfaces that can respond to the user’s input.
- Smarter and more elegant interfaces are being developed because of drag and drop features.
- Fast for real-time applications and if lots of simultaneous requests and responses needed in case of back-end use (Node.js).

Like every programmer gets and uses the full capabilities of programming languages, this thesis chose JavaScript projects from GitHub to test the Lehman's Laws for the above reasons that have been presented with lots of details.

## 2.3 Details about JavaScript Language and GitHub Software Development Platform

### 2.3.1 Details about JavaScript Language

JavaScript frequently shortened as JS first appeared on December 4 in 1995 which is 21 years ago [4]. It is a high-level dynamic, object-based, multi-paradigm, interpreted and weakly typed programming language. World Wide Web uses three core technologies which are JavaScript, HTML, CSS and that makes JavaScript a standard software language. The last stable version is ECMAScript June 2017 [5]. The primary use is to create web pages interactive and implement online programs, including video games. The bulk of websites operate it, and all current web browsers have a built-in JavaScript engine and support it without the need for any plug-ins.

Each of the multiple JavaScript engines serves a different implementation of JavaScript, all based on the ECMAScript spec, with some engines not supporting the specification thoroughly, and with many engines supporting extra features exceeding ECMA.

As a multi-paradigm language, JavaScript supports event-driven, useful, and imperative programming styles. The API helps JS to work with arrays, text, regular expressions, dates and necessary administration of the DOM, but it excludes any I/O, such as storage, networking, or graphics facilities, and uses for these the host environment in which it is embedded.

JavaScript engines are now embedded in numerous other types of host software, including in non-web software such as word processors and PDF program and also server-side in web servers and databases. Furthermore, in runtime conditions that make JavaScript accessible for writing desktop and mobile applications, including desktop widgets despite initially being only implemented in client-side web browsers.

Although there are apparent outer connections among JavaScript and Java, including language name, several standard libraries, and syntax, the two languages are different and vary considerably in design. Self and Scheme are two languages that JavaScript was influenced by [6].

### 2.3.2 Key Features of JavaScript

JavaScript also has some main features that make it a new and robust language. First of all, it is supported widely, and all common web browsers have built-in interpreters. JavaScript supports the structured programming syntax of other languages like C and makes a separation between statements and expressions. One change that helps the developers is that it allows omitting the semicolons (automatic semicolon insertion) [14].

Moreover, JavaScript is a dynamically typed language which means that a type is combined with a specific value and not only with every expression. If someone wants to present that with an example, it will be that variables can change types from a number to a string, etc. [15]

This language also provides run-time evaluation with the `eval()` function which can run statements in string format at run-time. JavaScript (JS) is almost utterly object-oriented like most object-based languages, but a difference is that it uses prototypes where other languages use classes for heritage [16].

Functions and methods don't differ at all like in other programming languages. Functions can be also defined as object constructors and have a double usage simultaneously with their essential role. A new function call to an existing older will produce an instance of a prototype with the heritage of properties and methods from the constructor [17]. One more significant feature is that functions are being considered as objects and could have properties, methods and could be first-class or nested functions with the lexical scope of the external functions [18]. If we are talking about the nested functions, the internal function object will be a segment of the outer function and inheritance things from it [19].

An unlimited amount of parameters can be transferred to a function. This is well-known as "Variadic" functions. The function can reach them through formal parameters or the local arguments object and create them with the `bind` method. Array and Object literals can be created with fast syntax commands which are also the foundation of the JSON data format. JS processes messages from a queue at a rate of one at a time and also creates a call stack frame which expands or retracts upon its needs. When this event loop happens, it allows the program's input or output to be performed. In more simple words the event loop doesn't block the other procedures, for example, a mouse click while simultaneously waiting for database queries to send back information [20]. In addition, JavaScript can have regular expressions which give a strong syntax for text administration [21]. Last but not least there are more features that some engines support and this thesis is going to present them in a referential manner:

- property getter and setter functions [22]
- conditional catch clauses
- iterator protocol
- shallow generators-coroutines
- array comprehensions and generator expressions
- proper block scope via the `let` keyword
- array and object destructuring
- concise function expressions
- ECMAScript for XML (E4X), an extension that adds native XML support to ECMAScript [23]



<b>JavaScript</b>	<b>PHP</b>
The JavaScript code is obtainable even after the output hasn't already produced.	The PHP code is accessible only after server performs the needing procedures.
JavaScript can manage certain local assignments.	PHP executes on servers, and the primary responsibility is to generate the HTML code that browsers interpret.
JavaScript can be mixed with XML, HTML, and AJAX.	PHP can be mixed with HTML and not XML.
JavaScript doesn't mainly use MySQL as a database but other types.	The central database of a PHP software is MySQL.
JavaScript runs in a browser.	PHP doesn't run in a browser but in a server, a compiler as a program or elsewhere.
With the URL of a file written in the address bar of a web browser, JavaScript can send files of accessible data.	PHP can get files from other web pages and also import them from the available server. The use of PHP is to create web pages with the power of a server.

*Table 6: JavaScript and PHP fundamental features differences*

### 2.3.3 GitHub Software Development Platform

GitHub is a web-based or version control repository and Internet hosting service founded on February 8 in 2008. The primary use of it is for code and offers all of the assigned version control and source code management (SCM) functionality like Git and also adds its specific characteristics. It provides access control and various collaboration features such as feature requests, bug tracking, task management, and wikis for every project developers add [7].

GitHub allows both methods for free and private repositories on the same account [8] which are usually used to host open-source software programs [9]. In April of this 2017, GitHub reports having approximately 57 million repositories and 20 million users, [10] making it the most prominent host of source code in the whole world [11].



## 3. ANALYSIS AND DESIGN OF THE WEB APPLICATION

In this chapter, there will be an analysis of the system that this thesis uses to get the results from the GitHub open-source projects. Initially, requirements will be described and technology that has been used. Afterwards, details of the programming language will be presented and after that information about the open-source programs that produce specific essential metrics. Moreover, the reader will be informed about the operating systems and more critical aspects of the code and also the functionalities. Lastly, the use of the entire system will be explained along with the processes to extract the results. In the end, two categories of data will be explained the more general, and the focused to releases of each project and details of the values that obtained.

### 3.1 Description of the Requirements

The platform has to retrieve information from GitHub by parsing the website and store them into a database. It also has to get data from an analysis and save them into the same database. The purpose is to get measurements for each official release of 100 JavaScript projects and keep the results of them. Then an analysis of the results will be performed, and conclusions will be drawn from it. The need of a combined system to collect all the data was essential. Note that this application is made for the creator's purpose to get the needing data or programmers that want to do a similar study of GitHub open-source applications.

### 3.2 Technology and Tools

#### 3.2.1 Operating Systems

A Windows 10 x64-bit personal computer has been used for the primary tasks like Database store of variables, SonarQube analysis and so on. This system was combined with a Raspberry Pi 3 model B that runs Raspbian [27] and had to run JSClassFinder to compute some extra metrics. JSClassFinder wasn't working at the Windows System so the Raspberry Pi was a good system that could handle the load of work.

#### 3.2.2 XAMPP – The Most Popular PHP Development Environment

XAMPP is a free open-source and cross-platform web server that this thesis used to gather the data locally to a computer. It is developed by Apache Friends [28]. XAMPP is the first letters of five packets well-known as Cross-Platform (X), Apache (A), MariaDB (M), PHP (P) and Perl (P). It is a lightweight Apache release that makes easy for programmers to create a local server like this thesis used. The main reason of use was for executing code that takes data from GitHub and other Analysis programs and also gathers all that into a Database. Everything needed to set

up a web server at a “localhost” computer is included. It works equally well on Linux, Mac, and Windows but the use was tested into a Windows system as mentioned in the 3.2.1 chapter. With this software, phpMyAdmin is added which is a free open-source managing tool for MySQL and MariaDB. It has become one of the most popular MySQL administration tools for web hosting services.

### **3.2.3 LARAVEL and MySQL**

Laravel is a free and open-source PHP web framework that makes easier the creation of code, maintenance and the transaction of the whole code to other systems if needed. It has lots of features like modular packaging system with a dedicated dependency manager, several ways for obtaining relational Databases, services that help in application deployment and very common and easy syntactic. It is also one of the most popular PHP frameworks of March 2015 and in more current statistics [29, 30].

MySQL is an open-source relational database management system [31] and has been used as a Database type to create as cell-format and store the data that are being processed afterward.

### **3.2.4 Goutte - A Simple PHP Web Scrapper**

It is a screen scraping and crawling web library for PHP that has been used to get general and more specific variables from the GitHub projects. It gives the power to crawl websites and extract data from the HTML responses. Moreover, with this tool, the scrapping of SonarQube Analysis is achievable using the same way that parses HTML web pages from GitHub. This a module that has been added to Laravel code and XAMPP web system [32].

### **3.2.5 Git – Version Control System**

Git is a version control system for tracking differences in computer files and coordinate work with other people. It is used for source code administration in software development [33]. It is very fast [34], keeps the data as it is [35] and also supports various workflows at the same time [36]. Furthermore, the use of it at this particular thesis is to download the Projects and change the release version just to be tested by SonarQube. The latest release is 6.5 that this thesis uses and has been published since 3<sup>rd</sup> of August.

### **3.2.6 SonarQube Continuous Code Quality**

SonarQube well-known also as Sonar [37] in previous versions is an open-source program that allows endless review of the state and quality that code has. The measurements that SonarQube provides are summed up in the following table:

Name	Description
Complexity	It is the complexity calculated based on the number of paths through the code. [66]
Cognitive Complexity	How hard it is to understand the code's control flow. [66]
Complexity / file	Average <b>complexity</b> by <b>file</b> .
Complexity / function	Average <b>complexity</b> by <b>function</b> .
Comment lines	The number of lines containing either comment or commented-out code.
Comments (%)	Density of comment lines = <b>Comment lines / (Lines of code + Comment lines) * 100</b>
Duplicated blocks	The number of duplicated blocks of lines.
Duplicated files	The number of files involved in duplications.
Duplicated lines	The number of lines involved in duplications.
Duplicated lines (%)	Density of duplication = <b>Duplicated lines / Lines * 100</b>
Technical Debt Ratio (Maintainability)	The ratio between the cost to develop the software and the cost to fix it.
Issues	The number of issues.
Code Smells	The number of code smells.
Bugs	The number of bugs.
Vulnerabilities	Number of vulnerabilities.
Functions	The number of functions.
Statements	The number of statements.
Files	The number of files.
Directories	The number of directories.
Lines	The number of physical lines.
Lines of Code	The number of physical lines that contain one or more characters which aren't whitespace or tabulation or part of a comment.

*Table 7: SonarQube Metrics*

It performs statistical analysis of code to find bugs; code smells, security vulnerabilities and other software problems. It is a multi-language tool because 20+ programming languages

including JavaScript are being covered for analyses. Furthermore, it can be used manually and independently and also to be integrated into other software like Eclipse, Visual Studio and more. It has a significant number of plugins [38, 39] to fulfill all users needs and provides additional metrics for example duplicated code, code coverage, code complexity, comments, etc. [40, 41]. SonarQube has Sonar-Scanner which is the tool that scans the GitHub projects and gets the analytics that later is being displayed to it. This means that SonarQube is a system that reports the results and gives the view option to users.

### 3.2.7 JSClassFinder – A Tool to Detect Class-Like Structures in JavaScript

This program is developed by a team and has two stages the first is preprocessing and the second is a visualization of the processed results. Moreover, the first step is qualified for the examination of the AST source code and the creation of object-oriented models which is a transformation of the initial code [42]. The second step is the access of the user to visualize and gain the results of the previous process and use the features that he or she might want. To execute this software you have firstly to run Pharo which is a complete environment for programming and running object-oriented codes. Like, JSClassFinder Pharo, has many features, for example, the basic are a live update, hot recompilation and administration [42]. The system requirements are:

- AST of a JavaScript source code in JSON format
- Pharo image with JSClassFinder

The results of this program are the following metrics:

- Total Number of Classes (NOC)
- Total Number of Attributes (NOM)
- Total Number of Methods (NOA)
- Total Number of Children (subclasses)
- The depth of Inheritance Tree (DIT).

Last but not least, it has a specific architecture which users have to follow in order metrics to be created and this can be described by the figure below:

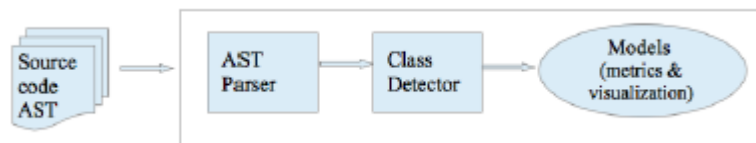


Figure 1 : JSClassFinder's architecture [42]

### 3.2.8 AutoHotkey – The Ultimate Automation Scripting Language for Windows

It is a free, open-source custom scripting language for Windows. The first use was to give an easy keyboard shortcuts or hotkeys access and software automation. With this tool data from GitHub Projects were gained because the parser can get HTML variables and store them in the Database. Furthermore, scripts were running and saving the complete web pages that were needing. So, AutoHotkey helped with the automation of repetitive tasks of the entire procedure.

### 3.2.9 Grafana – The Open Platform for Beautiful Analytics and Monitoring

The results are being presented directly from the Database with SQL queries with the use of Grafana. It is a leading open-source software for time series analytics which supports various types of Databases. In addition, there are 30 data sources, 27 panels, 16 apps and 461 dashboards available by the time this thesis is being written [43]. With this tool, the display of the results is pretty straightforward, accurate and nicely presented. Moreover, programmers can avoid the Excel graph format to present results which is more difficult. It also has the option to be hosted via the official website or run it manually. Of course, this thesis adapted the second option. Metrics are presented in five different ways depending on

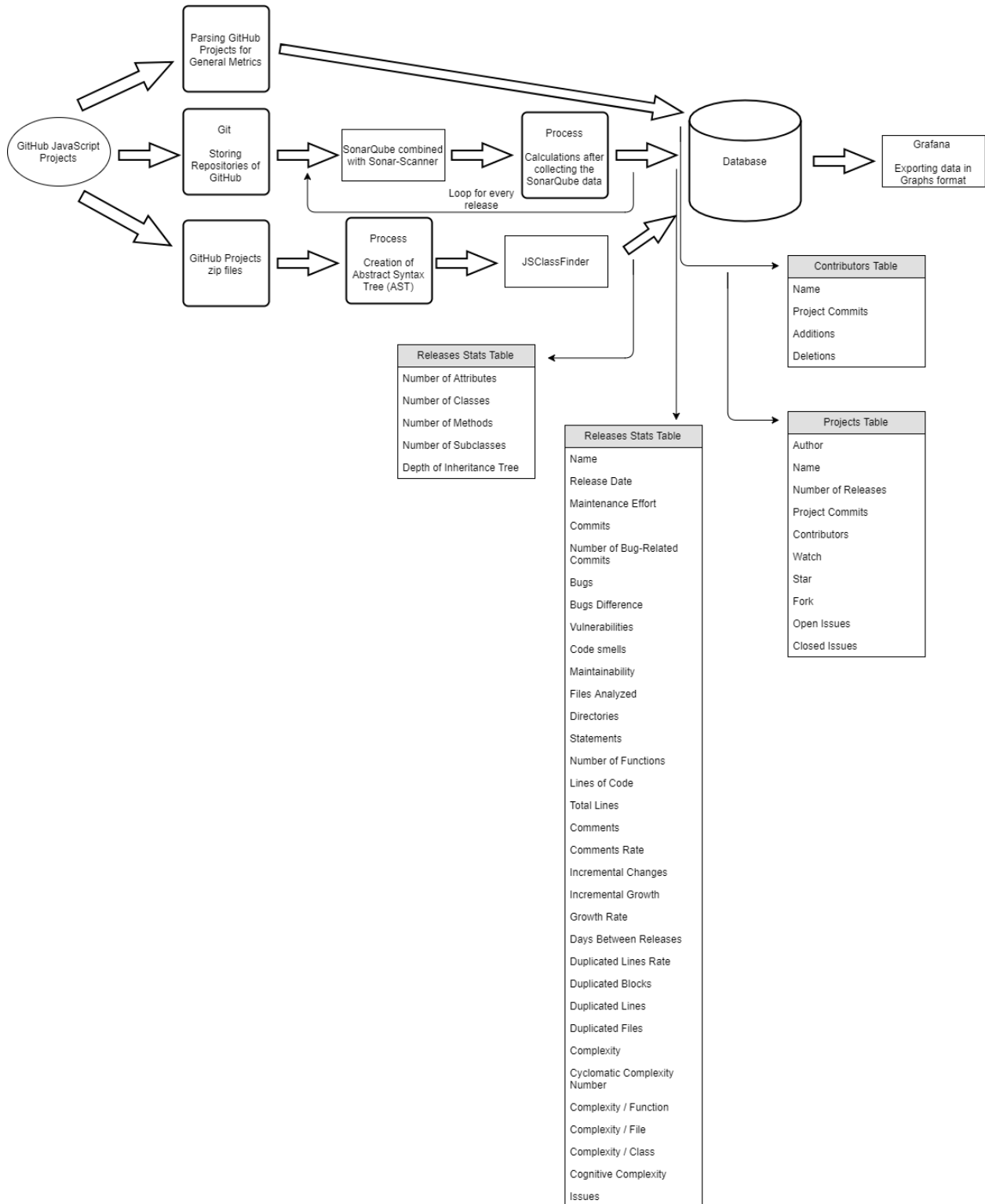
Name	Description
Time Graph	The x-axis represents time, and the data is grouped by time.
Series Graph	The data is grouped by series and not by time. The y-axis still represents the value.
Histogram	It is a kind of bar chart that groups numbers into areas, often called buckets or bins. Lower bars show that fewer data falls in that range.
Spark Lines	They are a great way of seeing the historical data related to the summary stat, providing valuable context at a glance.
Gauge	It gives a clear picture of how high value is in its context. The user can adjust and set the right thresholds for specific values. When a value exceeds the particular amount that we mentioned before the color will change.

*Table 8: Presentation of results*

### 3.3 The JS evolution Tool

In this chapter, the use of the specific software that has been created to receive measurements is going to be analyzed. So, let's start with a general use-case of all the programs that this thesis uses. Firstly, the setup procedure is going to be explained and after that the use-case of the software to get metrics for one Project of the total number which is 100. Moreover, the next figure shows in detail the entire process that is used to extract data from the combined JS evolution tool:

## Software Evolution



*Figure 2: JS tool combined usage*

## Software Evolution

The XAMPP has to be downloaded [44], installed and opened with the Apache and MySQL running (Figure 3). Then the SonarQube must be downloaded [45] from the official website and has to be executed. To achieve that the user has to run StartSonar Batch File by double-clicking it (Figure 3).

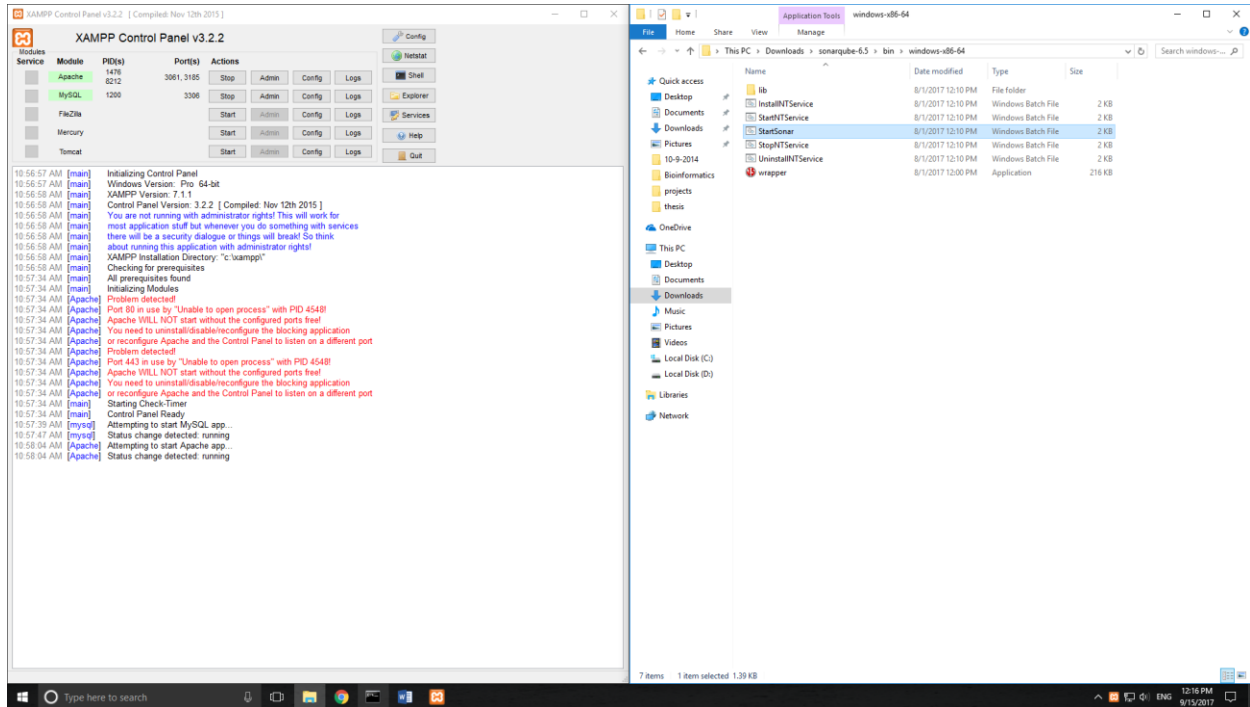


Figure 3: Starting XAMPP and SonarQube

The thesis folder of the entire project copy must be stored in the following path like this: "C:\xampp\htdocs\thesis" (Figure 4).



## Software Evolution

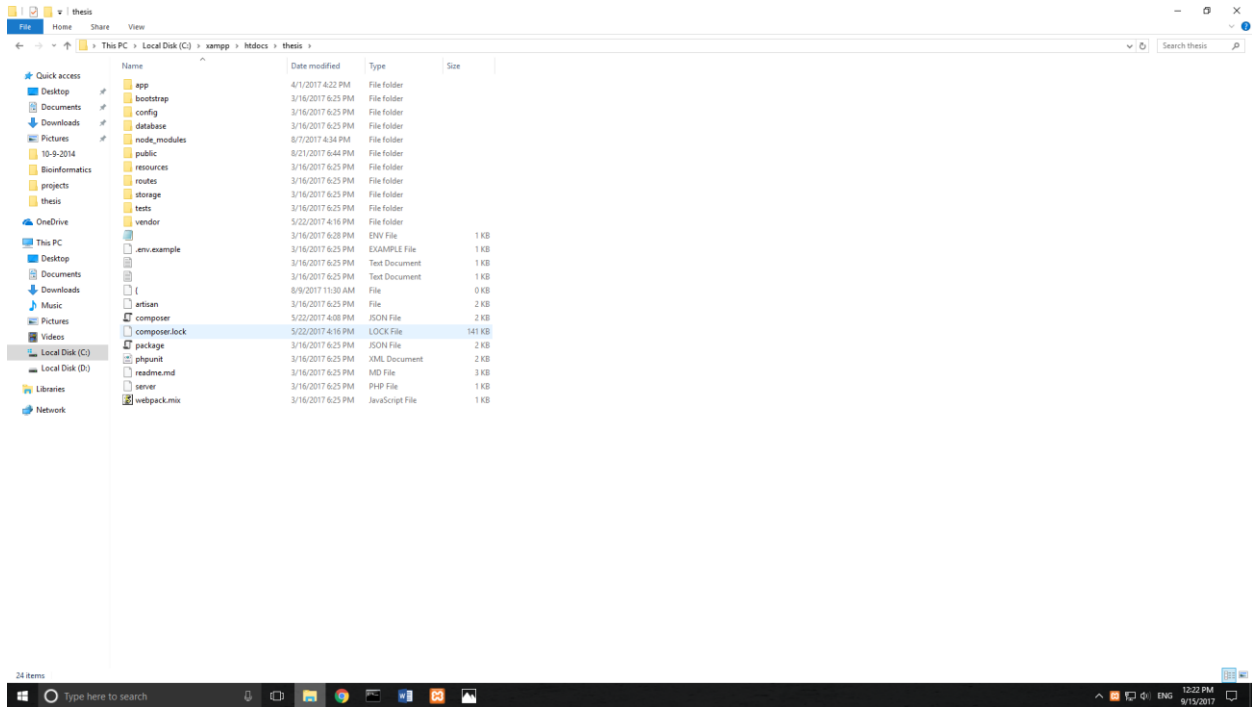


Figure 4: The entire thesis folder

To get the projects into the local computer, the user has to download [46], install and run the Git (Git Bash) and also go to the GitHub web page to choose the Projects. In this thesis, the JavaScript language combined with sort by “Most forks” has been selected (Figure 5). The place where Git has been installed and the user has to download Projects is “C:\Users\”Username”\git.” The commands to Git clone a repository can be shown in the following picture (Figure 5):



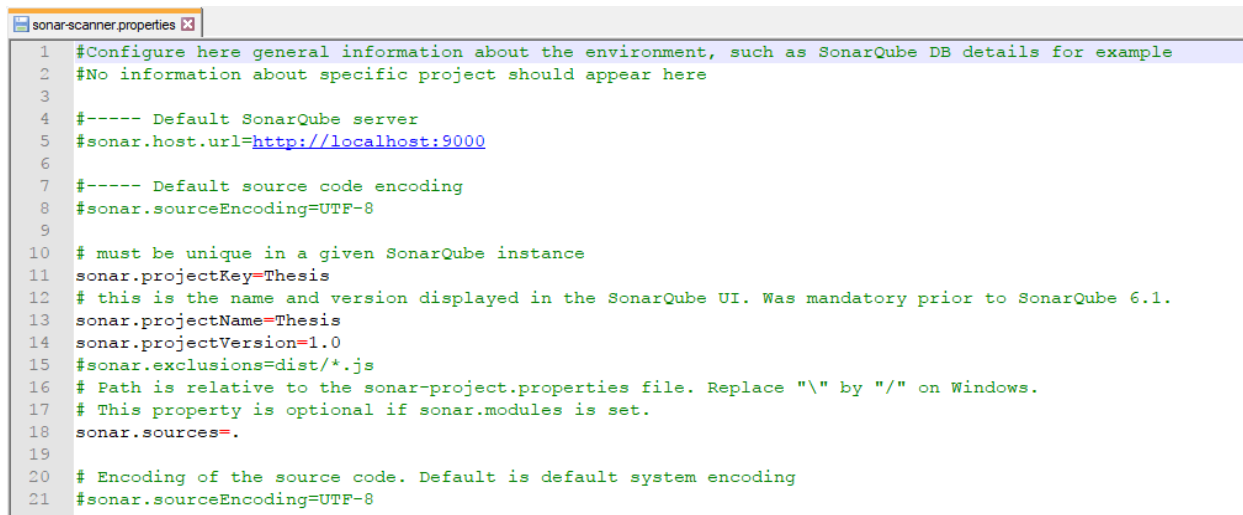
```

MINGW64:/c/Users/Aggelos/.git
Angelos@Angelos MINGW64 ~ (master)
$ cd .git
Angelos@Angelos MINGW64 ~/.git (master)
$ git clone https://github.com/bower/bower
Cloning into 'bower'...
remote: Counting objects: 15907, done.
remote: Compressing objects: 100% (29/29), done.
remote: Total 15907 (delta 4), reused 16 (delta 3), pack-reused 15875
Receiving objects: 100% (15907/15907), 4.05 MiB | 130.00 KiB/s, done.
Resolving deltas: 100% (8176/8176), done.
Angelos@Angelos MINGW64 ~/.git (master)
$

```

Figure 5: Git clone of a repository

To analyze and present the results to the SonarQube service which is running on localhost:9000 when you start the StartSonar Batch File the user has to download the SonarQube Scanner [47] and store it at “C:\Users\“Username”\Downloads.” SonarQube Scanner has a sonar-scanner.properties file stored in “conf” folder in which some parameters have to be synchronized with SonarQube. For example the keys of the project same as SonarQube like the above screenshot (Figure 6):



```

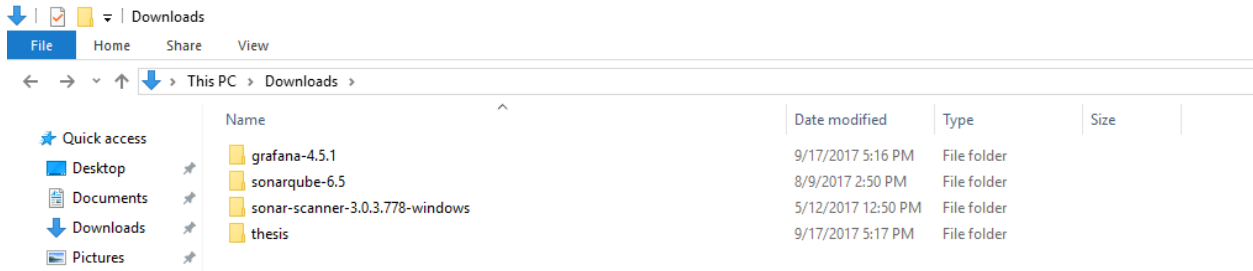
sonar-scanner.properties
1 #Configure here general information about the environment, such as SonarQube DB details for example
2 #No information about specific project should appear here
3
4 #----- Default SonarQube server
5 #sonar.host.url=http://localhost:9000
6
7 #----- Default source code encoding
8 #sonar.sourceEncoding=UTF-8
9
10 # must be unique in a given SonarQube instance
11 sonar.projectKey=Thesis
12 # this is the name and version displayed in the SonarQube UI. Was mandatory prior to SonarQube 6.1.
13 sonar.projectName=Thesis
14 sonar.projectVersion=1.0
15 #sonar.exclusions=dist/*.js
16 # Path is relative to the sonar-project.properties file. Replace "\" by "/" on Windows.
17 # This property is optional if sonar.modules is set.
18 sonar.sources=.
19
20 # Encoding of the source code. Default is default system encoding
21 #sonar.sourceEncoding=UTF-8

```

Figure 6: SonarQube Scanner configurations

If for any reason the tester needs to close the SonarQube there is a StopNTService Batch File that he or she has to execute to achieve that. Sometimes java.exe must stop to restart and run

the software again. In this case, you can use the command “taskkill /f /im java.exe” to terminate java.exe processes that might run in the background and then start SonarQube. The above programs should be like the following screenshot (Figure 7):



*Figure 7: Four main folders necessary for the analysis*

Moreover, the AutoHotkey has to be download [48] and installed along with a Desktop shortcut and the two scripts for making procedures more automatic. To be more specific the following screenshot shows the three archives needed to be on the Desktop of the Computer (Figure 8):



*Figure 8: AutoHotkey and two necessary scripts*

Furthermore, a ready-to-use Pharo image with JSClassFinder installed [49] is also essential to get the other metrics of NOA, NOC, NOM, NOS, and DIT.

Last but not least we have to download and set up Grafana [50]. Because XAMPP needs port 3000 and Grafana also, we can change the Grafana’s port to 8000 at configurations file.

## 4. PLATFORM PRESENTATION

The web application has five main web addresses (URLs). URL stands for Uniform Resource Locator and is a reference to a web resource that specifies its location on a computer network.

- <http://localhost/thesis/public/projects/create/{id}>
- <http://localhost/thesis/public/projects/manualSQ/{id}>
- <http://localhost/thesis/public/projects/calculations/{id}>
- <http://localhost/thesis/public/projects/contributors/{id}>
- <http://localhost/thesis/public/projects/JSClassFinder/{id}>

Every URL has its use in the platform. More details are presented in the following chapters from 4.1 to 4.5.

### 4.1 Creation of the Project and Basic Metrics with the Automated Procedure

First of all, there is a core web page that the software uses to create some vital things as the Project itself and also simultaneously to get the number of releases (Figure 9).

	id	author	name	number_rel	proj_commits	contributors	watch	star	fork	open_issues	closed_issues
	25	webpack	webpack	253	4545	343	1240	31779	3986	498	3764
	15	innovate	mean	1885	179	657	10301	3117	188	188	963
	28	enyo	dropzone	97	787	53	425	12852	3057	677	734
	8	kriasoft	react-starter-kit	8	846	125	550	15332	3137	348	522
	30	Leaflet	Leaflet	36	6364	517	851	19395	3298	254	3236
	31	emberjs	ember.js	259	14866	680	1046	18229	3752	246	5109
	32	angular-ui	ui-grid	78	3808	298	403	4916	2472	1460	3407
	33	hexojs	hexo	120	2386	102	731	10113	2678	264	2167
	34	pixijs	pixi.js	79	4447	245	818	15710	2668	310	2380
	35	lodash	lodash	380	7859	239	706	26172	2726	0	2603
	36	jquery-validation	jquery-validation	17	782	175	480	8040	2402	134	1247
	37	RocketChat	Rocket Chat	101	12160	335	706	13859	2891	1058	3596
	38	vuex	vuex	34	690	159	461	9619	2839	31	542
	39	Modernizr	Modernizr	27	2399	222	1027	21084	2641	146	1107
	40	hammerjs	hammer.js	25	0	0	0	0	0	0	0

Figure 9: Projects table in Database

For example at the “<http://localhost/thesis/public/projects/create/{id}>” the tester can create a project by adding the id to the URL of the following identifier in the Database that has not been

established so far. Also, the user has to complete the Author, Name, and Number of Releases of the exact Project on GitHub (Figure 10).

localhost/thesis/public/p x

localhost/thesis/public/projects/create/42

## New Project

Author

Name

Number of Releases

Figure 10: Main thesis functionality through creation of a new Project

Then, it creates the precise release that the SonarQube is testing and initialize the values to zero. For example, the last version is shown in the following screenshot (Figure 11):

id	project_id	name	rel_date	maintenance_effort	commits	NOBug_related_commits	bugs	bugs_differ	vulnerabilities	code_smells	maintainability	files_analyzed	directories	statements	NOF
3001	39	v2010.07.06dev	2010-07-16	0.000	2205	0.000	272	0	21	3066	2.500	33	13	33107	6952
3002	39	v1.1	2009-12-07	0.000	2127	0.000	272	0	21	3096	2.500	33	13	33090	6944
3003	40	v2.0.0	2016-04-22	0.000	53	0.000	82	0	9	1599	3.800	137	16	23014	3722
3004	40	v2.0.7	2016-04-21	0.000	66	0.000	0	0	0	0	0.000	0	0	0	0

Figure 11: For every release stats and metrics in Database

## 4.2 Generating Basic Metrics with the Manual Procedure

Sometimes the power of a personal computer isn't enough for this kind of Projects, and the automated procedure fails to store into the Database the metrics that will after be used for the evaluation of the Software Evolution. In addition, to generate the variables for testing the validity of Lehman's Laws as the 4.3 chapter mentions a manual procedure has to be done. When lots of zeros are presented to the Database, this means something went wrong and the execution of SonarQube manually is essential. For example, the release with id 1285 has been skipped (Figure 12).

Showing rows 1275 - 1299 (3852 total. Query took 0.0055 seconds.) [project\_id: 13... - 14...][rel\_date: 2013-09-27... - 2016-01-25...]

SELECT \* FROM `releases\_stats` ORDER BY `project\_id` ASC, `rel\_date` DESC

Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

<< < 52 > >> Number of rows: 25 Filter rows: Search this table Sort by key: releases\_stats\_project\_id\_index (ASC)

Options		id	project_id	name	rel_date	maintenance_effort	commits	NOBug_related_commits	bugs	bugs_differ	vulnerabilities	code_smells	maintainability	files_analyzed	directories	statements	NOF
	1276	13	0.3.1	2013-09-27	0.000	6537	0.000	15	0	1	1113	2.300	110	35	13421	2955	
	1277	13	0.3.0	2013-09-27	0.000	6595	0.000	15	0	1	1113	2.300	110	35	13304	2917	
	1278	13	dashboard	2013-09-12	0.000	6815	0.000	15	0	1	1072	2.400	112	36	12446	2765	
	1279	13	0.2.1	2013-08-07	0.000	7113	0.000	15	0	1	1285	2.900	94	34	13627	2713	
	1280	13	0.2.0	2013-07-11	0.000	7212	0.000	10	0	1	893	3.200	71	28	8263	1529	
	1281	13	0.1.1	2013-06-24	0.000	7323	0.000	386	0	10	1606	5.300	71	29	10492	2071	
	1282	14	2.9.4	2017-03-10	0.000	33	0.000	25	0	4	246	2.300	14	6	4374	648	
	1283	14	2.9.3	2017-03-01	0.000	51	0.000	25	0	4	240	2.300	14	6	4350	638	
	1284	14	2.9.2	2016-12-19	0.000	76	0.000	25	0	4	240	2.300	14	6	4348	638	
	1285	14	2.9.1.1	2016-12-19	0.000	78	0.000	0	0	0	0	0.000	0	0	0	0	

Figure 12: Zeros in a line in Database at 1285 identifier and version name 2.9.1.1

The Project id is 14, so if the user goes to Projects table, he or she can determine the author and name of the Project. In this case, author name is "alvarotrigo" and name of the Project "fullPage.js." Then the tester has to find the exact tag of the version in a coded format which is "eac6956" in our case. The website will be <https://github.com/author name/project name/releases/tag/version name> and in our example <https://github.com/alvarotrigo/fullPage.js/releases/tag/2.9.1.1> (Figure 13).

## Software Evolution

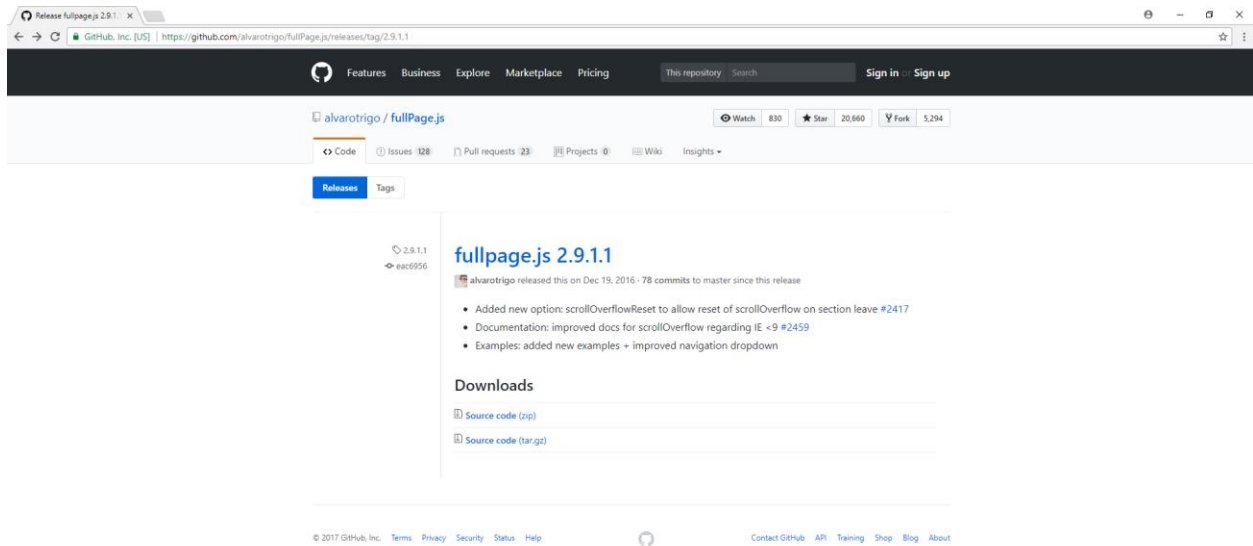


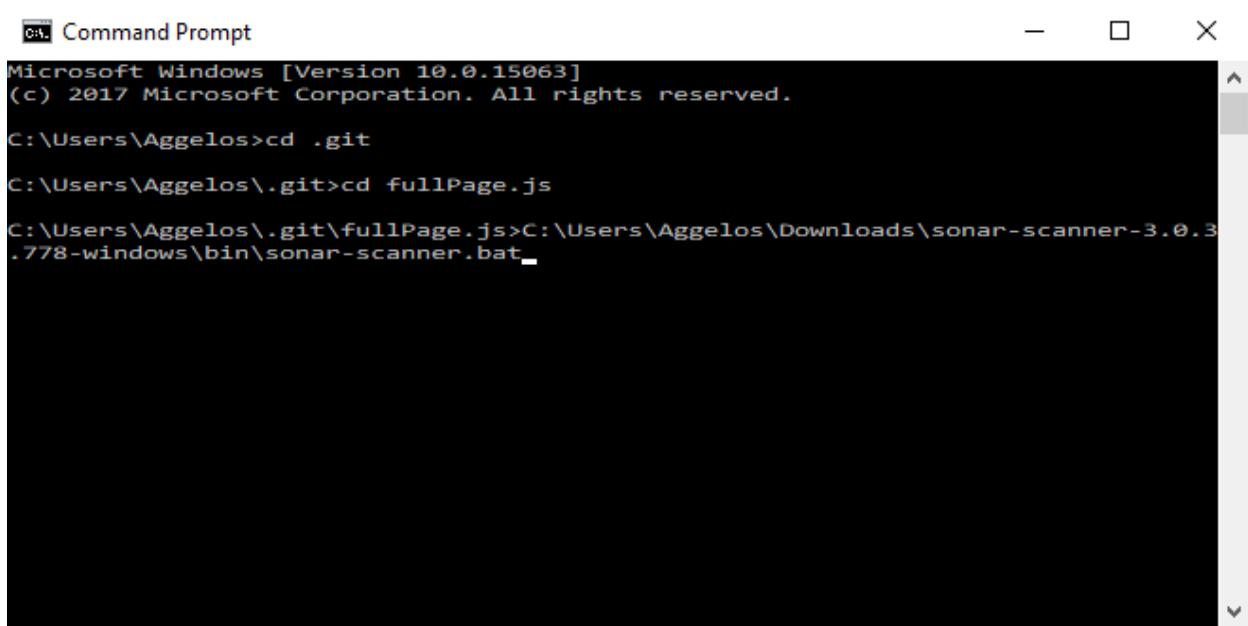
Figure 13: Specific's release tag from GitHub official website

Afterwards, the encoded tag must be set to the folder with the command “git reset –hard “tag”” for the exact Project name (Figure 14). The location of the folder is known from the 3.3.1 chapter.

```
MINGW64:/c:/Users/Aggelos/.git/fullPage.js
Angelos@Angelos MINGW64 ~ (master)
$ cd .git
Angelos@Angelos MINGW64 ~/.git (master)
$ cd fullPage.js/
Angelos@Angelos MINGW64 ~/.git/fullPage.js (master)
$ git reset --hard eac6956
HEAD is now at eac6956 Merge pull request #2467 from alvarotrigo/dev
Angelos@Angelos MINGW64 ~/.git/fullPage.js (master)
$
```

Figure 14: Setting release's tag manually to Git for SonarQube Scanner

To run and start the scanning of a release the user already set before with the tag the same location must be accessed through a command prompt. And then  
C:\Users\“Username”\Downloads\sonar-scanner\3.0.3.778-windows\bin\sonar-scanner.bat  
path must be executed from the first location mentioned before (Figure 15).



```
Command Prompt
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Aggelos>cd .git
C:\Users\Aggelos\.git>cd fullPage.js
C:\Users\Aggelos\.git\fullPage.js>C:\Users\Aggelos\Downloads\sonar-scanner-3.0.3.778-windows\bin\sonar-scanner.bat
```

*Figure 15: Manual execution of SonarQube Scanner*

When the “Execution Success” message appears the tester has to browse at <http://localhost:9000/component/measures?id=Thesis> (Figure ) and save the page as a complete web page at the C:\Users\“Username”\Downloads\thesis with the name “Measures – Thesis.”



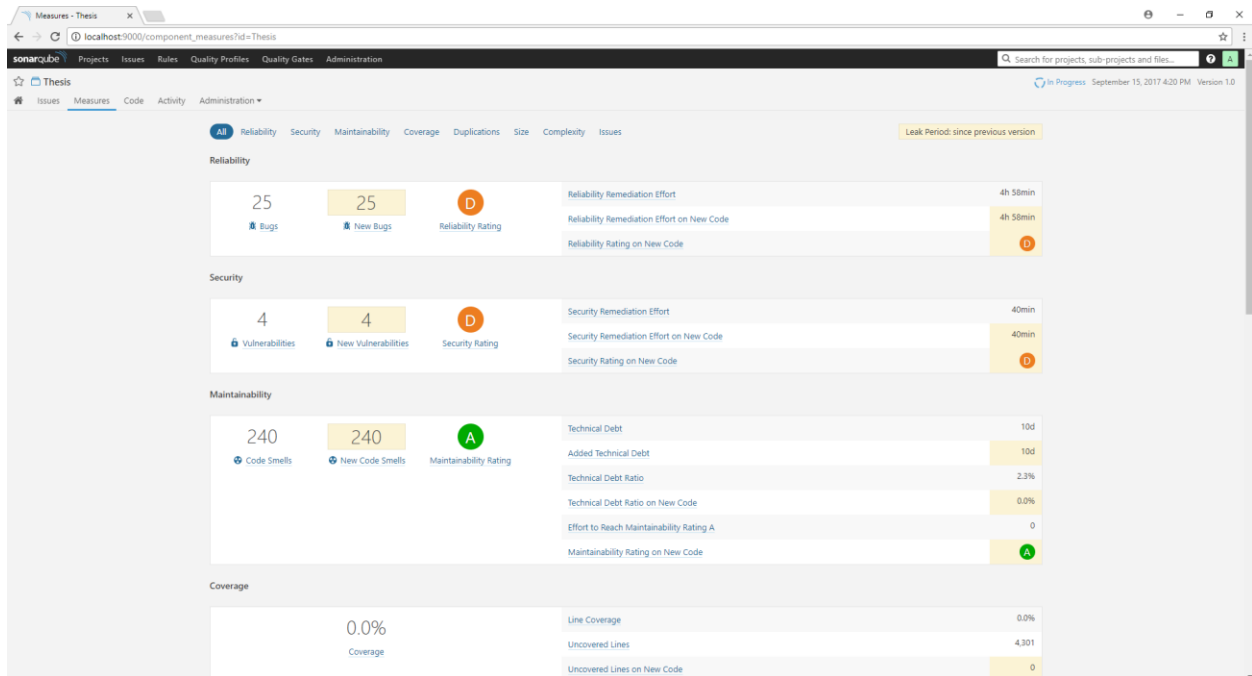


Figure 16: Save results from SonarQube server to HTML format

Lastly, the following page must be used to store the data from the web page to the Database. The last variable in the URL is modified in relation to the id that user tried to fix (Figure 17).

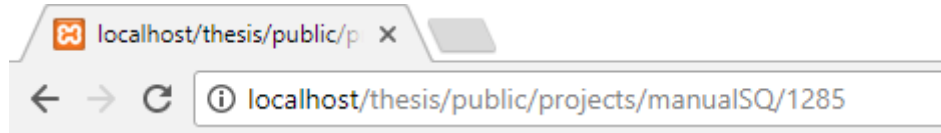


Figure 17: Store results in the Database for a specific identifier

### 4.3 Calculations from the Existing Measurements

After finishing with the above two chapters and tester has everything that is being needed, some calculations have to be done to find the measurements that are going to verify the Software Evolution of open-source Programs. More details about these metrics are on 4.9 chapter. The screenshot (Figure 18) shows the execution of the exact URL starting from id = 1 because we want this counts from the first Project stored in the Database. A similar procedure is being used for the contributors at the next chapter (4.4).



Figure 18: Perform calculations for all the releases

#### 4.4 Receiving the Contributors and Stats of Them for Every Project

The general metrics for the Project are being saved to the Database from the 4.1 automated procedure and as another form which are the Contributors of the Project. The maximum available stats for Contributors could be 100 or less. By running the Figure's URL (Figure 19), the contributors are being stored to the Database one by one for every Project the project's table has information. It is an automated procedure and similar to 4.4 chapter.

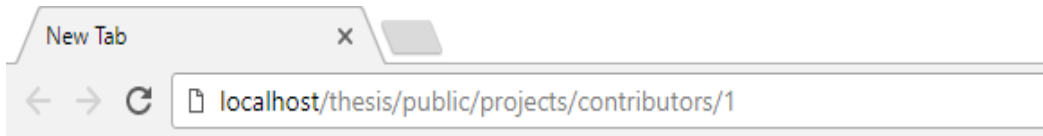


Figure 19: Store the maximum of 100 contributors in the Database

#### 4.5 JSClassFinder Variables Added to the Thesis Database

Raspberry Pi 3 model B has been used for taking some additional metrics that 3.2.7 chapter mentions with every detail. For every project and release, we set the path to the version the tester wants to accumulate the excess metrics. To do that he or she has to place after "/s" (source) parameter the path "C:\Users\"Username"\Downloads\thesis\"Release name\"\*.js". The "\*" means that the tester needs every folder and subfolder of this release that has an ending of .js (JavaScript files). The "/y" means to accept all the files to be copied to the text file that the user defines later with the path "C:\Users\"Username"\Desktop\"File name.txt" (Figure 30).

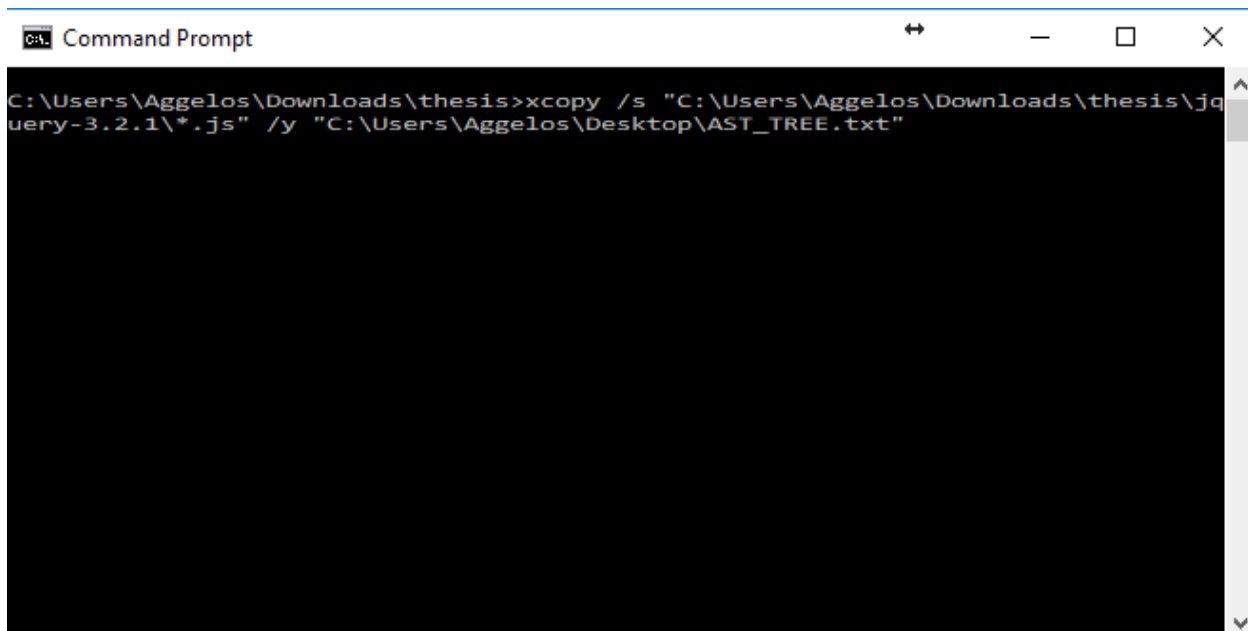


Figure 20: Generate a file from every JavaScript file in a release

To continue with the process, there is a web page called <http://esprima.org/demo/parse.html> in which the user has to copy the content of the text file, for example, AST\_TREE.txt we have here to the left column of the site. And then the right side will generate the AST tree (Abstract syntax tree) of the JavaScript code we gave at the first time (Figure 31).

The screenshot shows a web browser window with the URL <http://esprima.org/demo/parse.html>. The page title is "Esprima Parser" and the subtitle is "Parser produces the (beautiful) syntax tree". The left pane shows the JavaScript code from the file AST\_TREE.txt, which includes a function `wrap` and a test function `testWrap`. The right pane shows the generated AST tree, which is a JSON object representing the abstract syntax tree of the code. The tree structure is as follows:

```

{
  "type": "Program",
  "body": [
    {
      "type": "ExpressionStatement",
      "expression": {
        "type": "CallExpression",
        "callee": {
          "type": "FunctionExpression",
          "id": null,
          "params": [],
          "body": {
            "type": "BlockStatement",
            "body": [
              {
                "type": "IfStatement",
                "test": {
                  "type": "UnaryExpression",
                  "operator": "!",
                  "argument": {
                    "type": "MemberExpression",
                    "computed": false,
                    "object": {
                      "type": "This",
                      "name": "this"
                    },
                    "property": {

```

Figure 21: Parse the generic file and export the AST tree

Afterwards, the content of the right column on this website must be copied to a new folder with the name "ast.json" or whatever the tester wants it to be called (Figure 32). Because in this thesis an external use of a Raspberry Pi 3 model B as another computer has been used it is necessary to transfer this file with WinSCP at the "home/pi/pharo" path (Figure 33).

## Software Evolution

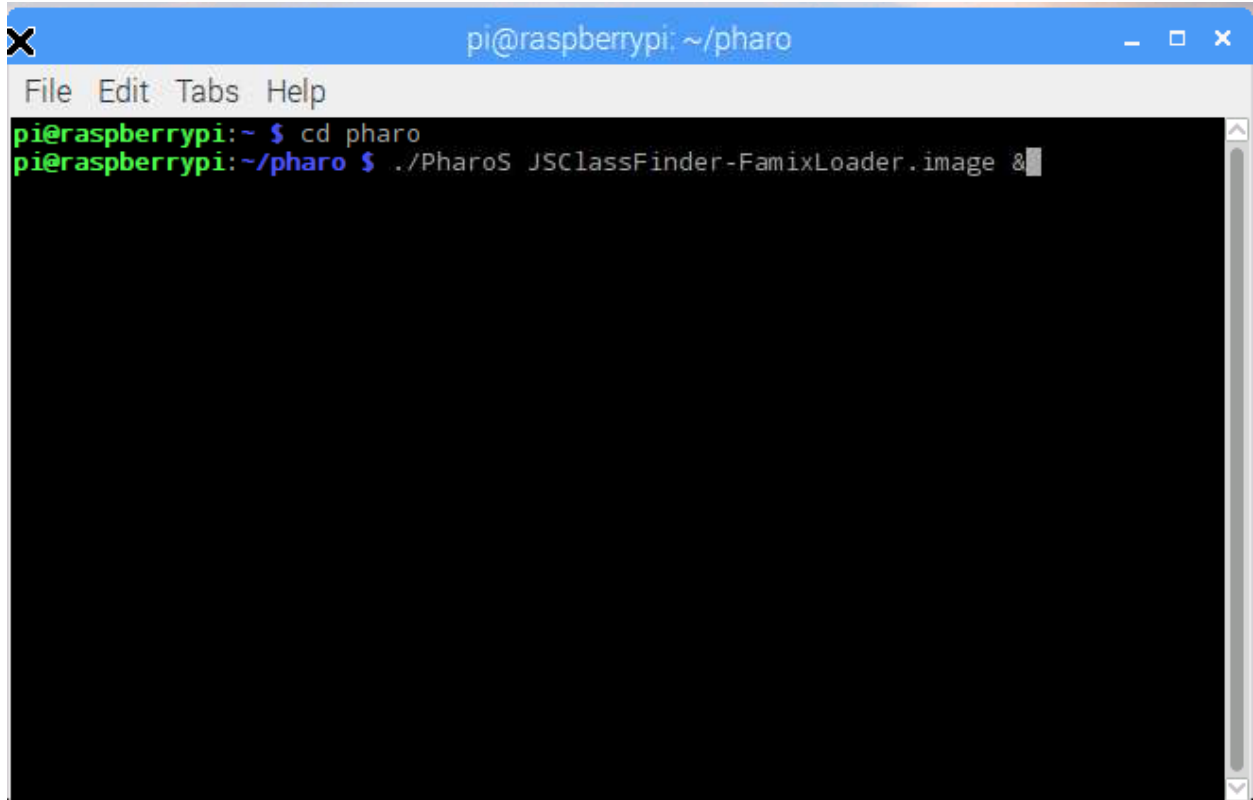
```
9607 { "type": "ExpressionStatement",
9608   "expression": {
9609     "type": "CallExpression",
9610     "callee": {
9611       "type": "MemberExpression",
9612       "computed": false,
9613       "object": {
9614         "type": "CallExpression",
9615         "callee": {
9616           "type": "Identifier",
9617           "name": "jQuery"
9618         },
9619         "arguments": [
9620           {
9621             "type": "Identifier",
9622             "name": "script"
9623           }
9624         ],
9625         "property": {
9626           "type": "Identifier",
9627           "name": "remove"
9628         }
9629       },
9630       "arguments": []
9631     },
9632     "generator": false,
9633     "expression": false,
9634     "async": false
9635   },
9636   "sourceType": "script"
9637 },
9638 { "type": "BlockStatement",
9639   "body": [
9640     {
9641       "type": "ExpressionStatement",
9642       "expression": {
9643         "type": "CallExpression",
9644         "callee": {
9645           "type": "FunctionExpression",
9646           "left": null,
9647           "params": [
9648             {
9649               "type": "BlockStatement",
9650               "body": [
9651                 {
9652                   "type": "IfStatement",
9653                   "test": {
9654                     "type": "UnaryExpression",
9655                     "operator": "!",
9656                     "argument": {
9657                       "type": "MemberExpression",
9658                       "computed": false,
9659                       "object": {
9660                         "type": "Identifier",
9661                         "name": "script"
9662                       },
9663                       "property": {
9664                         "type": "Identifier",
9665                         "name": "wrap"
9666                       }
9667                     }
9668                   }
9669                 }
9670               ]
9671             }
9672           ]
9673         }
9674       }
9675     }
9676   ]
9677 }
```

Figure 22: Copy and paste the AST tree to a JSON file

Name	Size	Changed	Rights
image.Nuyd3r		9/17/2017 12:58:52 PM	rw-r--r--
play-cache		8/16/2017 8:17:00 PM	rw-r--r--
play-stash		8/8/2017 4:29:54 AM	rw-r--r--
ast.json	765 KB	9/17/2017 12:57:04 PM	rw-r--r--
JSClassFinder-FamixL...	27,734 KB	9/17/2017 12:40:15 PM	rw-r--r--
JSClassFinder-FamixL...	158,673 KB	8/16/2017 3:54:27 PM	rw-r--r--
libB3DAcceleratorPlu...	93 KB	10/30/2014 3:05:30 PM	rw-rw-r--
libInternetConfigPlug...	14 KB	10/30/2014 3:06:04 PM	rw-rw-r--
libJPEGReaderPlugin.so	37 KB	10/30/2014 3:05:30 PM	rw-rw-r--

Figure 23: Send JSON file to JSClassFinder

Then to execute the JSClassFinder software, Pharo must be running. To do so (Figure 34) we get to the installation folder and run the JSClassFinder image that was embedded to Pharo.

A terminal window titled 'pi@raspberrypi: ~/pharo' with a menu bar containing 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows two lines of command execution: 'pi@raspberrypi:~ \$ cd pharo' and 'pi@raspberrypi:~/pharo \$ ./PharoS JSClassFinder-FamixLoader.image &'. The rest of the terminal area is black with a white cursor at the end of the second line.

```
pi@raspberrypi:~ $ cd pharo
pi@raspberrypi:~/pharo $ ./PharoS JSClassFinder-FamixLoader.image &
```

*Figure 24: Execution of JSClassFinder image and Pharo*

Moreover, the tester has to load the .json file that previously was put on the right spot. He or she has to set a Name to determine the results and write only the file name for example "ast" without the .json ending (Figure 35).

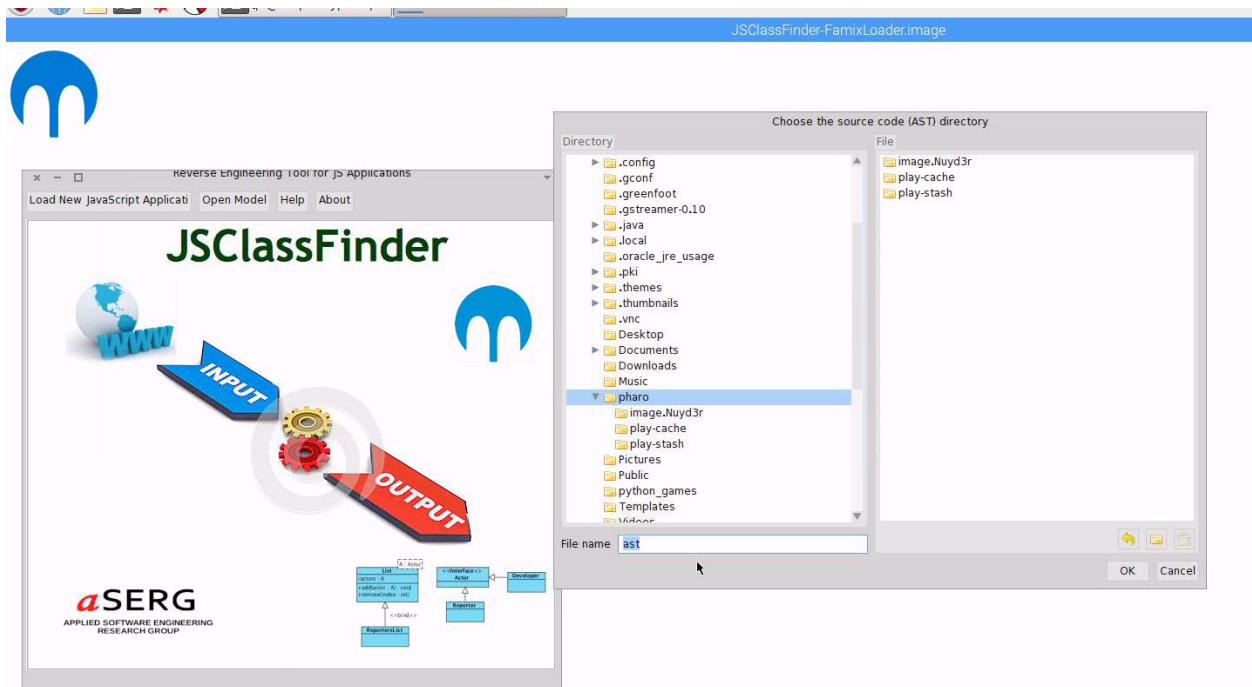


Figure 25: Run ast file for analysis

After a while, the results will be appeared (Figure 26), and we need them stored in the Database. To achieve that there is a URL with the name “JSClassFinder” in which everyone can put the specific identifier (id) and set these variables (Figure 28). At the same time, we have to compute the incremental changes. We go to every release web page, for example, <https://github.com/hexojs/hexo/compare/3.2.1...master> and search with Ctrl+F the word “function” (Figure 27). Then we store the result at the “incremental changes” field. The system will multiply this value by a 40% because some of these functions are not truly changed but added or deleted. The last number the system get will be added to the “incremental growth” value, and that will eventually be the “incremental changes” (Figure 28).

## Software Evolution

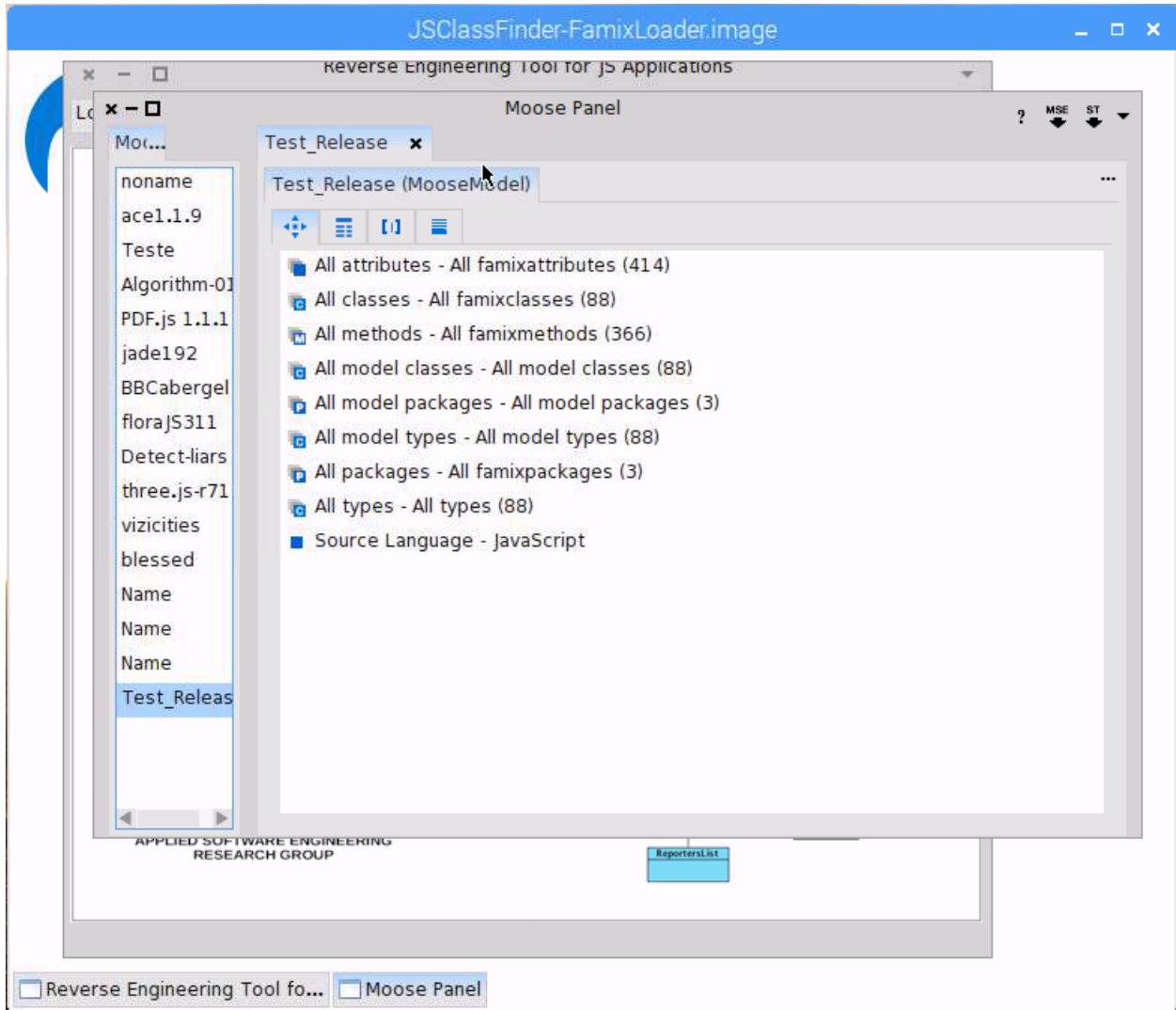


Figure 26: Results from JSClassFinder software

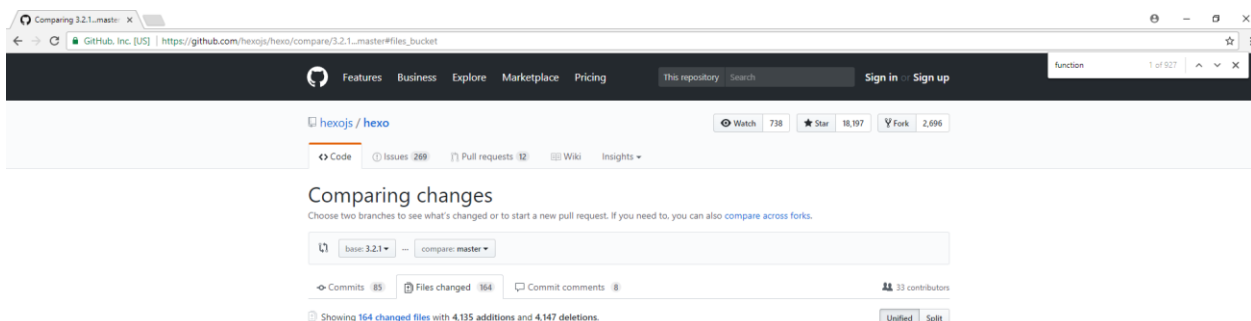
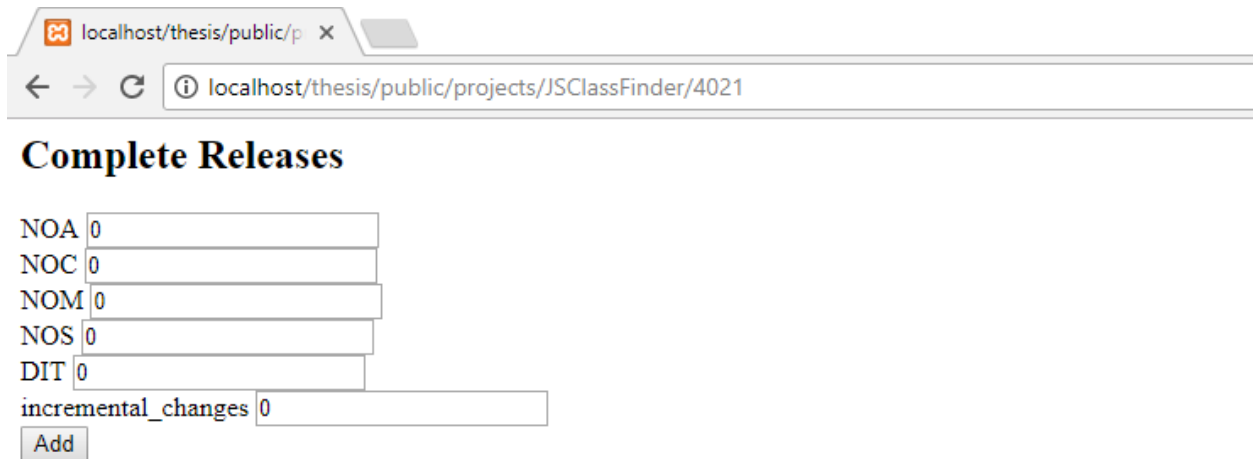


Figure 27: Search for "function" key-word in GitHub for every release



The screenshot shows a web browser window with the address bar displaying 'localhost/thesis/public/projects/JSClassFinder/4021'. Below the browser, there is a form titled 'Complete Releases'. The form contains six input fields, each with the value '0' entered: NOA, NOC, NOM, NOS, DIT, and incremental\_changes. An 'Add' button is located below the 'incremental\_changes' field.

*Figure 28: Store the results in the Database*

Note that the identifier (id) changes automatically every time the user adds the above measurements to the Database.

#### 4.6 General Fixes to Common Problems

Two are the most common problems that may occur. The first one is the automated saving procedure to get stuck because of insufficient CPU power to make the whole SonarQube analysis and run the scripts. To fix that the tester has to save and replace the file HTML file manually only at this point and then everything will run fine (Figure 29). Note that the SonarQube needs empty browser tabs to run flawlessly. So, all computer resource must be focused on the task of analysis. The second is to store zeros to the Database like the initial state and to check if it is right or wrong decision of the system, the user has to execute a manual analysis as chapter 4.2 mentioned before.



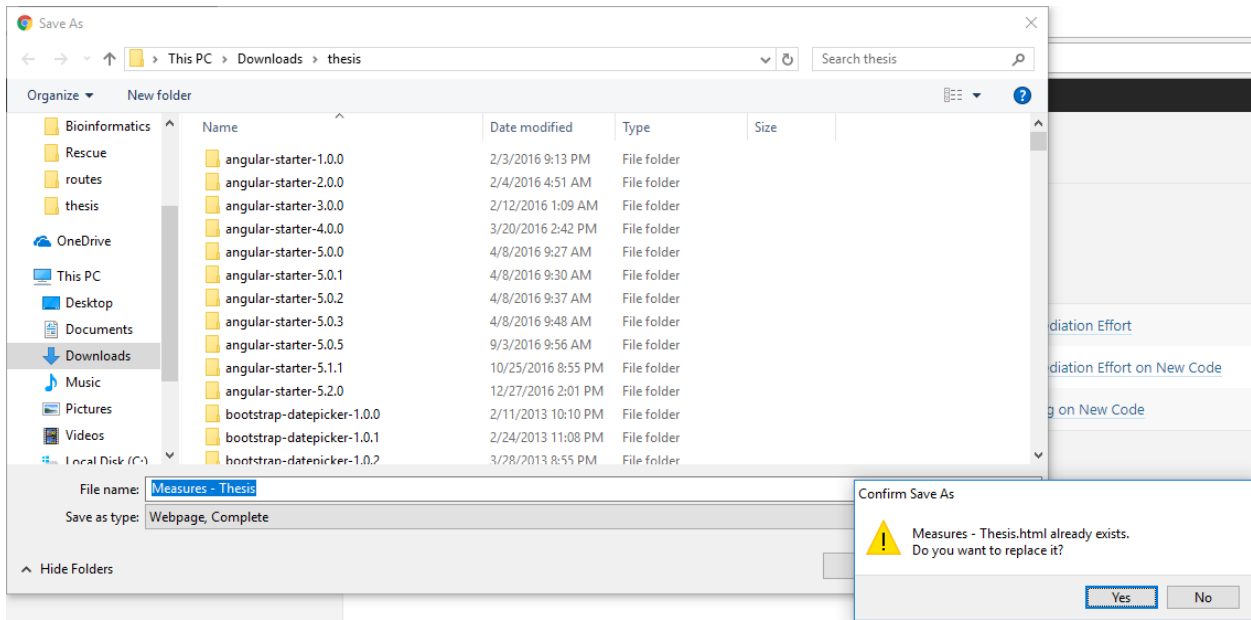


Figure 29: Fixing an underlying problem of automated procedure

#### 4.7 Creation of Charts with the Use of Grafana

After installing and setting up Grafana, we execute “grafana-server” file to localhost:8000 and find our Database (Figure 30). After that, we go to dashboards and create Graphs using SQL queries (Figure 31). Each Graph is being fixed to show the metrics the user already had stored like in this thesis. The chapter 3, in general, presents all the results and conclusions from the charts that Grafana generated. Grafana is the best way to show the results of this survey in a user-friendly form.

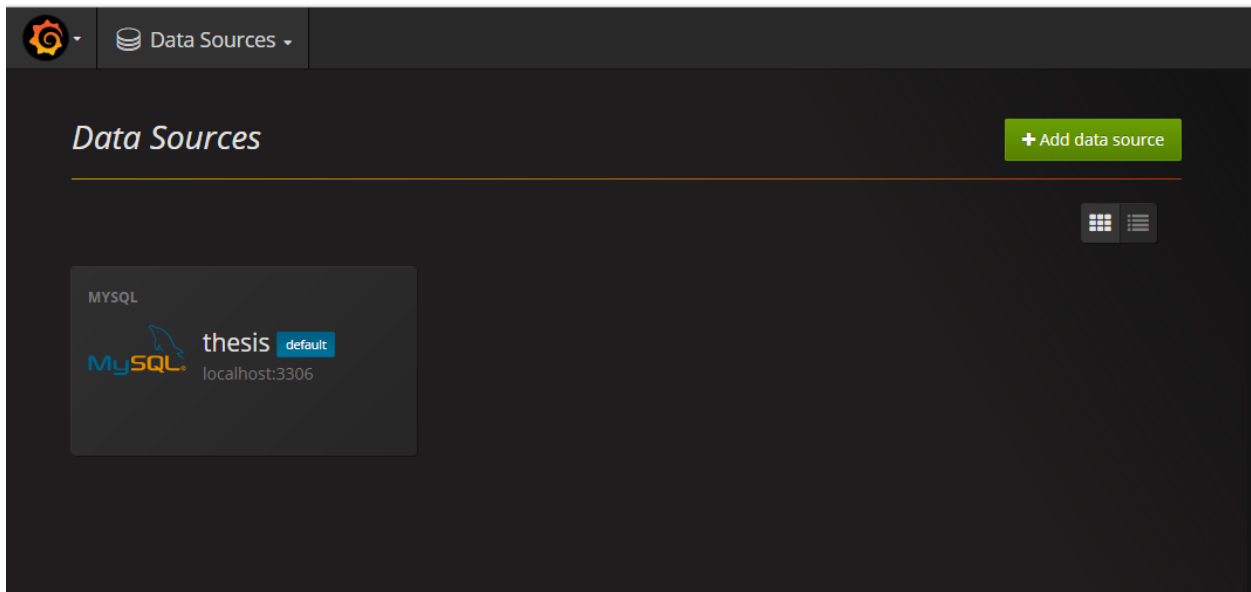


Figure 30: Connection to Database and Grafana

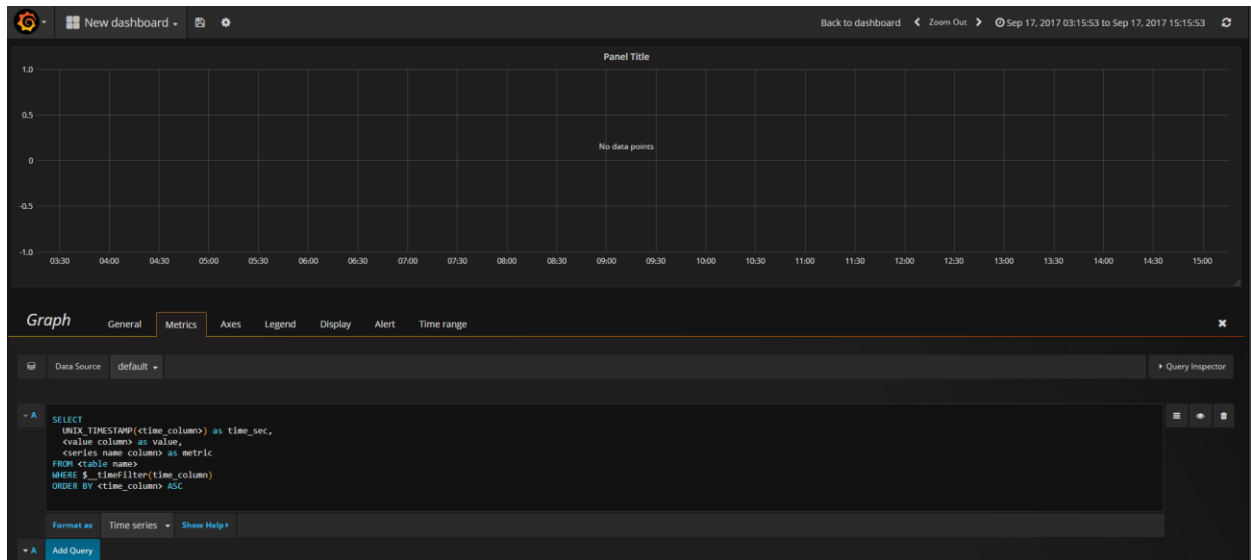


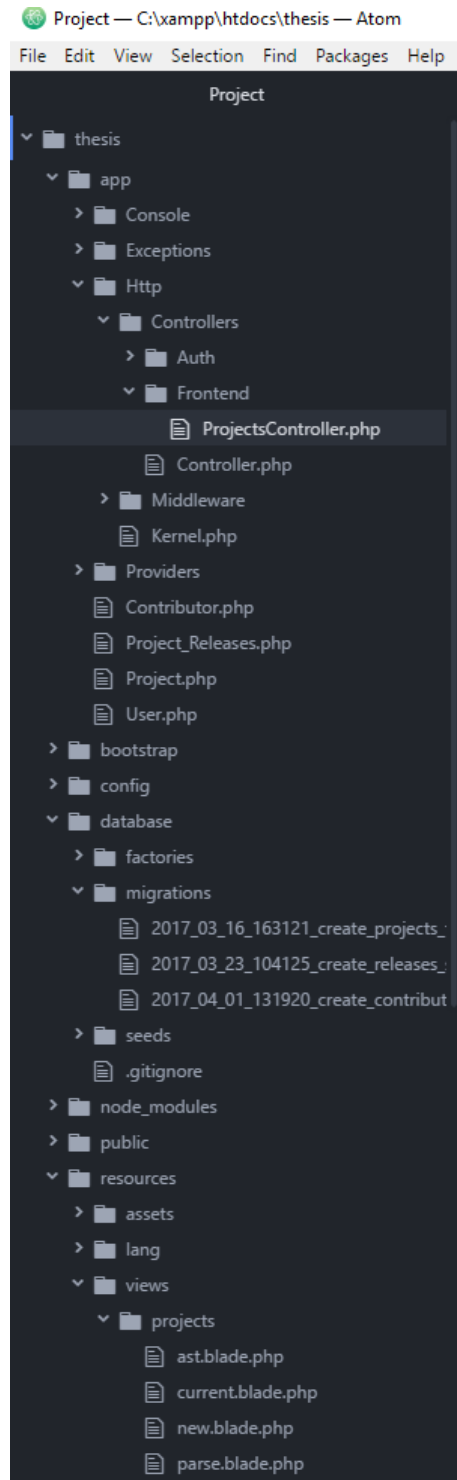
Figure 31: Creation of charts via SQL queries

## 4.8 Important Parts of Code

There are comments at the entire code, but in this chapter, the critical parts will be explained. The main file is ProjectsController where all the functionality is there. There are two big loops one for every page on GitHub that has ten releases and the second for the last page in which

## Software Evolution

releases number may vary. The essential parts are in the following sectors of the thesis project (Figure 32):



*Figure 32: Paths of basic code segments*

An uncertain number of GitHub projects has different commits URL. To get data referring to commits, the tester has to change the “...develop” line to the right one for the exact project (Figure 33). For example, some endings are “...latest,” “...dev,” etc.

```

$crawler = $client->request('GET', 'https://github.com/'.$author.'/'.$name.'/compare/'.$releases_names[$i].'...master'); // The basic URL for taking the number of commits and other data
//
$release_1_commits = $crawler->filter('span[class="num text-emphasized"]')->each(function ($node) {
    return $node->text();
});
$release_2_commits = $crawler->filter('span[class="counter"]')->each(function ($node) {
    return $node->text();
});
$release_3_commits = $crawler->filter('span[class="Counter"]')->each(function ($node) {
    return $node->text();
});
$formatted_date[$i] = date('Y-m-d', strtotime($relative_time[$i]));
$releases_stats = new Project_Releases();
$releases_stats->name = $releases_names[$i];
$releases_stats->project_id = $project->id;
if (empty($release_1_commits[0])) {
    if (empty($release_2_commits[3])) {
        if (empty($release_3_commits[3])) {
            $crawler = $client->request('GET', 'https://github.com/'.$author.'/'.$name.'/compare/'.$releases_names[$i].'...develop'); // Change this in case of different commits URL...
        }
    }
}
    
```

Figure 33: Part of code that might change for some GitHub projects

Furthermore, there is a script that allows the software to download zip files for all releases to get tested later with the JSClassFinder (Figure 34) as chapter 4.5 explained before.

```

<script>
    @if ( $temp != 0 )
    var win = @for ($j = 0; $j<$temp; $j++) @for ($i = 0; $i < 10; $i++)
    window.open('{!! $uris[$i][$j] !!}', '_blank')
    @endfor
    @endfor;
    win.focus();
    @endif
    var win_last = @for ($i = 0; $i < $temp2; $i++)
    window.open('{!! $uris[$i][$temp_last] !!}', '_blank')
    @endfor;
    win_last.focus();
</script>
    
```

Figure 34: Download of zip file for every release

### 4.9 Details of the Values (Metrics) Obtained

The following tables have every detail for all the metrics that this thesis used. It also contains calculations and more general information of each one of them. The first table is about the fundamental metrics for every release of each project. Then at the second and third tables, there are details for the general variables of projects.

Metric	Calculation	Description
--------	-------------	-------------

## Software Evolution

Name of Every Release (name)	-	For every release, names are being stored.
Release Date (rel_date)	-	For every release, the release date is being stored.
Maintenance Effort (maintenance_effort)	Incremental Changes / Days Between Releases	The effort of programmers to change and update the project.
Commits (commits)	-	The number of commits.
Number of Bug-Related Commits (NOBug_related_commits)	Bugs Difference*(-1) / Commits	The number of commits that are being associated with the solution of some bugs.
Bugs (bugs)	-	The number of bugs.
Bugs Difference	bugs[recent_release] – bugs[previous_release]	For every release bugs, the previous release bugs are being subtracted. The initial state of these calculations is the newest release.
Vulnerabilities (vulnerabilities)	-	Number of vulnerabilities.
Code smells (code_smells)	-	Number of code smells.
Maintainability (maintainability)	Remediation cost / (Cost to develop 1 line of code * Number of lines of code) [66]	Remediation cost: The cost to fix each issue from the rule is the same, and the formula is: The total remediation cost per file = number of issues x constant. The ratio between the cost to develop the software and the cost to fix it. The value of the cost to develop a line of code is 0.06 days. [66]
Files Analyzed (files_analyzed)	-	The number of files.
Directories (directories)	-	The number of directories.
Statements (statements)	-	The number of statements.
Number of Functions (NOF)	-	The number of functions.
Number of Attributes (NOA)	-	The number of attributes.

## Software Evolution

Number of Classes (NOC)	-	The number of classes (including nested classes, interfaces, enums and annotations).
Number of Methods (NOM)	-	The number of methods.
Number of Subclasses (NOS)	-	The number of children (subclasses).
Depth of Inheritance Tree (DIT)	-	Is defined as “the maximum length from the node to the root of the tree” [67]
Lines of Code (LOC)	-	The number of physical lines that contain at least one character which is neither a whitespace or a tabulation or part of a comment. [66]
Total Lines (Total_lines)	-	The number of physical lines.
Comments (comments)	Total Lines (Total_lines) – Lines of Code (LOC)	The number of lines containing either comment or commented-out code.  Non-significant comment lines (empty comment lines, comment lines containing only special characters, etc.) do not increase the number of comment lines. [66]
Comments Rate (comments_rate)	Density of comment lines = $\frac{\text{Comments}}{\text{Lines of code} + \text{Comments}} * 100$ [66]]	With such a formula: <ul style="list-style-type: none"> <li>• 50% means that the number of lines of code equals the number of comment lines.</li> <li>• 100% means that the file only contains comment lines. [66]</li> </ul>
Incremental Changes (incremental_changes)	Functions Added or Removed + Functions Modified	The number of functions that have been added removed and modified in total.
Incremental Growth (incremental_growth)	Functions Added or Removed	The number of functions that have been added and removed in total.
Growth Rate (growth_rate)	Functions Added or Removed / Days Between Releases	The number of functions that have been added and removed in total divided by Days Between Releases.

## Software Evolution

Days Between Releases (dif_days)	$\text{rel\_date}[\text{recent\_release}] - \text{rel\_date}[\text{previous\_release}]$	For every release date, I subtract the previous release date. The initial state of these calculations is the newest release.
Duplicated Lines Rate (duplicated_lines_rate)	Density of duplication = <b>Duplicated Lines</b> / Total Lines * 100	It is the % ratio of Duplicated Lines divided by Total Lines.
Duplicated Blocks (duplicated_blocks)	-	<p>The number of duplicated blocks of lines.</p> <p>For a block of code to be considered as duplicated:</p> <ul style="list-style-type: none"> <li>• There should be at least 100 successive and duplicated tokens.</li> <li>• Those tokens should be spread at least on ten lines of code.</li> </ul> <p>Differences in indentation as well as in string literals are ignored while detecting duplications. [66]</p>
Duplicated Lines (duplicated_lines)	-	The number of lines involved in duplications.
Duplicated Files (duplicated_files)	-	The number of files involved in duplications.
Complexity (complexity)	Cyclomatic Complexity Number / Lines of Code (CCN/LOC)	The Cyclomatic Complexity Number divided by the Lines of Code.
Cyclomatic Complexity Number (CCN)	-	It is the complexity calculated based on the number of paths through the code. Whenever the control flow of a function splits, the complexity counter gets incremented by one. Each <b>function</b> has a minimum complexity of 1. [66]
Cyclomatic Complexity Number / Function (complexity_function)	-	Average <b>complexity</b> by <b>function</b> .
Cyclomatic Complexity Number / File (complexity_file)	-	Average <b>complexity</b> by <b>file</b> .

Cyclomatic Complexity Number / Class (complexity_class)	Cyclomatic Complexity Number / Number of Classes (CCN / NOC)	Average <b>complexity</b> by class.
Cognitive Complexity (cognitive_complexity)	-	How hard it is to understand the code's control flow. [66]
Issues (issues)	-	SonarQube raises an issue every time a piece of code breaks a coding rule.

*Table 9 (Releases Stats Table): Metrics used to this thesis for validation of each Lehman law to every GitHub JavaScript program*

Metric	Description
Author (author)	The author name of the GitHub Project.
Name (name)	The name of the GitHub Project.
Number of Releases (number_rel)	The number of releases of a GitHub Project so far.
Project Commits (proj_commits)	How many commits every Project has in total.
Contributors (contributors)	The number of contributors this Project has.
Watch (watch)	The number of people watching a repository and want to get notifications from it.
Star (star)	This is the number of tracking Projects repositories that people submitted.
Fork (fork)	Is the number of copies of a repository that allows doing experiments with it. It also offers the possibility to change the path of the primary reason that a Project has been created for.
Open Issues (open_issues)	How many issues are being submitted and don't have a solution.
Closed Issues (closed_issues)	The number of resolved Issues that developers found a solution.

*Table 10 (Projects Table): General statistics metrics for every Project that has been tested*

Name (name)	This is the name of the contributor.
Project Commits (proj_commits)	The number of commits has every contributor in every Project.
Additions (add)	How many added parts of the basic code



## Software Evolution

	had been done.
Deletions (delet)	How many deleted parts of the basic code has been done.

*Table 11 (**Contributors Table**): Top 100 contributors to a Project with most commits*

## 5. RESULTS AND CONCLUSIONS

In this chapter, the results are going to be presented. The first part will have general information on how to test the results we already have collected. Afterwards, general metrics for the 100 projects will be presented with the help of Grafana charts. To continue with, for each one of the Lehman's laws of evolution, we have a table with results from the Mann-Kendall trend test [64] that we have already executed. Furthermore, we are going to draw conclusions about the Software Evolution of 100 projects and understanding whether JavaScript programs comply with the laws of evolution. Tables show results for 40 of the 100 projects but despite that analysis has performed to all the samples. The last part of this section after the validation of the laws is a sort comparison between JavaScript and other programming languages findings referring to evolution.

### 5.1 General Stats for JS projects

In the seven following graphs, we present general statistics about the projects we analyzed. The 1<sup>st</sup> figure out of 7 displays the "Number of Releases" for these 100 projects. Most of our programs have at least 50 official versions which make the research more reliable. The 2<sup>nd</sup> shows the "Number of Commits" in total for every project. As we can notice JavaScript programs have thousands of commits by their developers. They fix, update, improve and maintain the software on a regular basis. The 3<sup>rd</sup> and 5<sup>th</sup> figures indicate the popularity of the GitHub projects we tested. In addition, the 4<sup>th</sup> figure which is about the different forks (changed versions) of a program is the filter that we used in GitHub. That's why the charts are going from the higher value to the lowest. In the end, we have the last two graphs that are for the open and closed issues of a project which are indicating the effort of programmers to respond to customer's questions and meet their expectations in future updates.

# Software Evolution

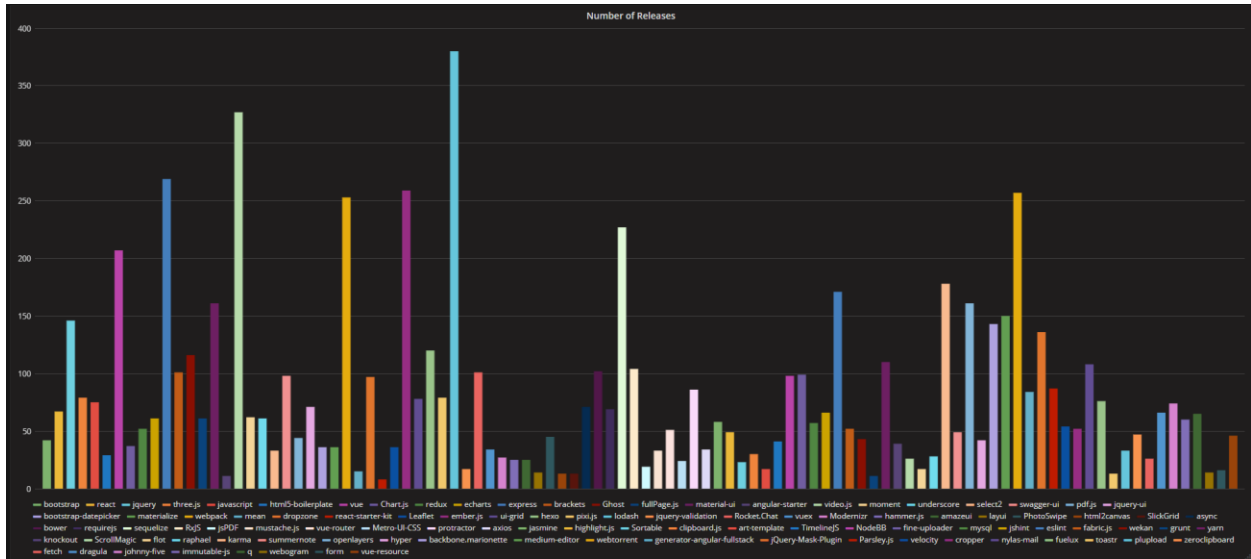


Figure 35: Number of releases for each project

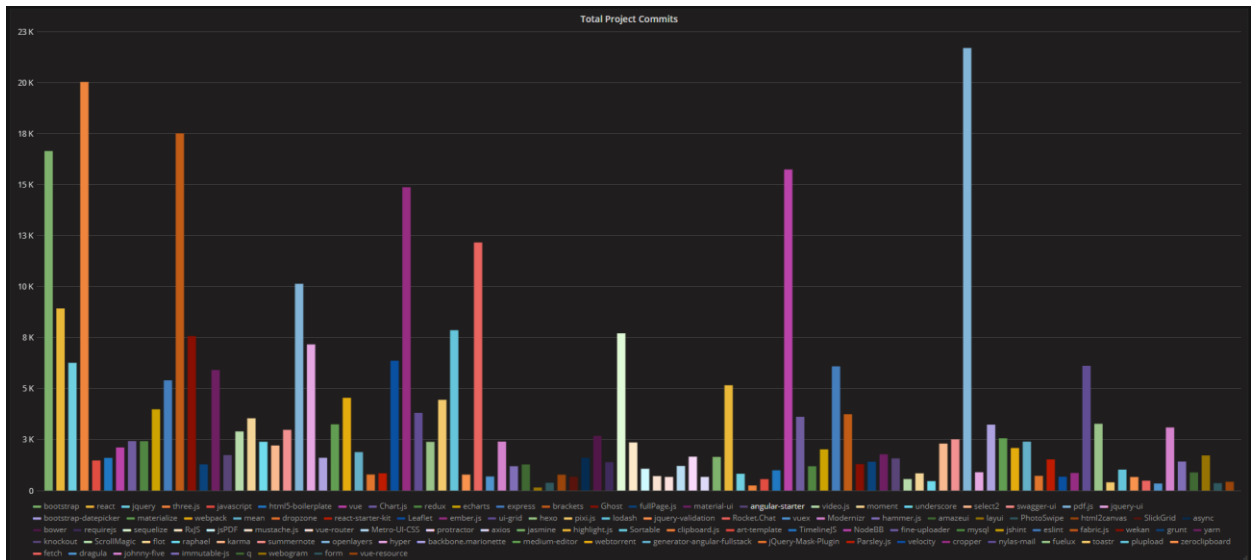


Figure 36: Number of commits for each project

# Software Evolution

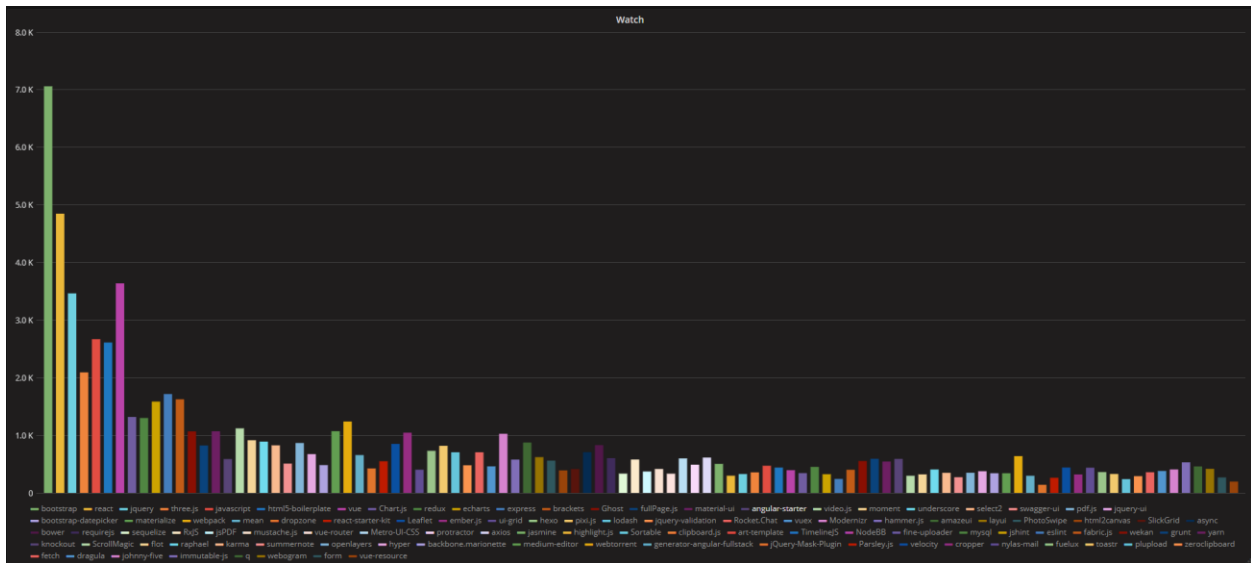


Figure 37: Number of watches for each project

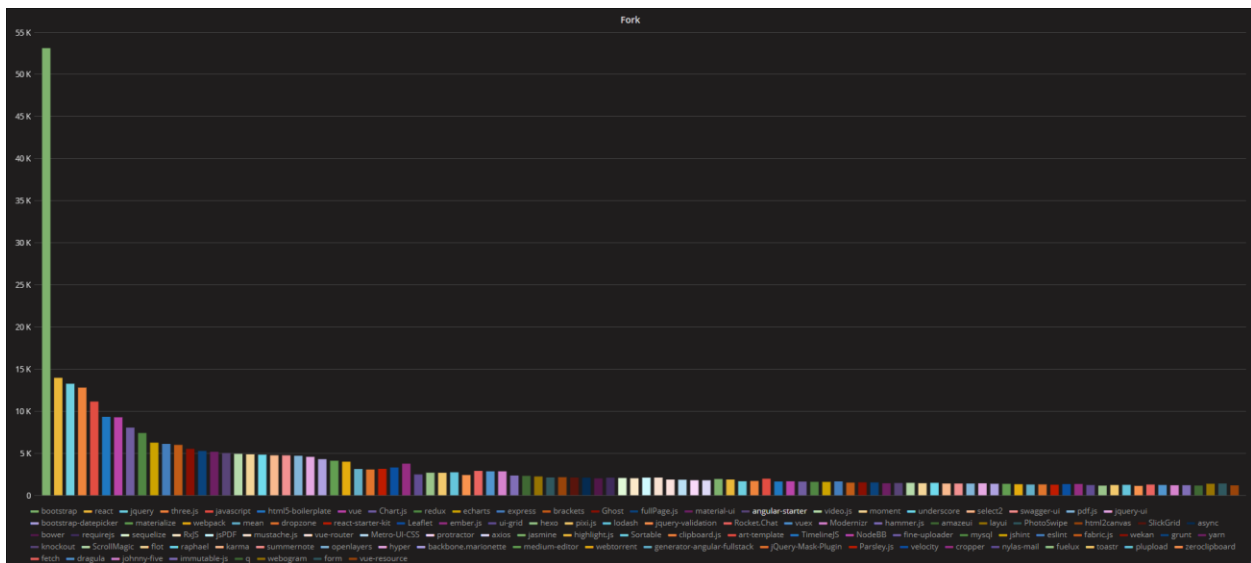


Figure 38: Number of forks created for each project

# Software Evolution

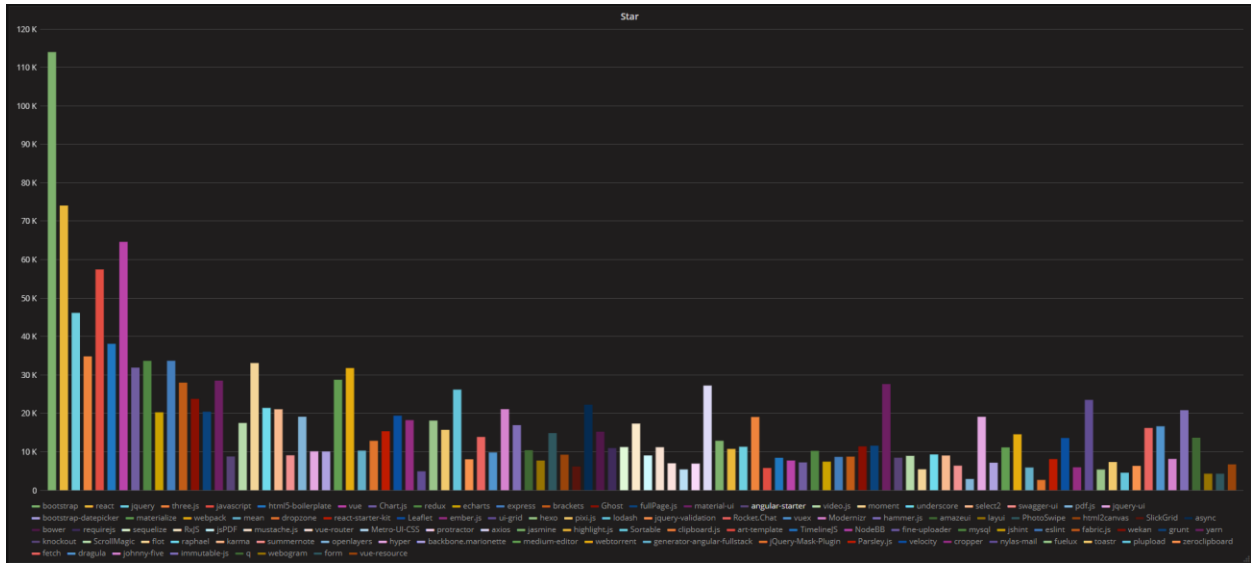


Figure 39: Number of Stars for each project

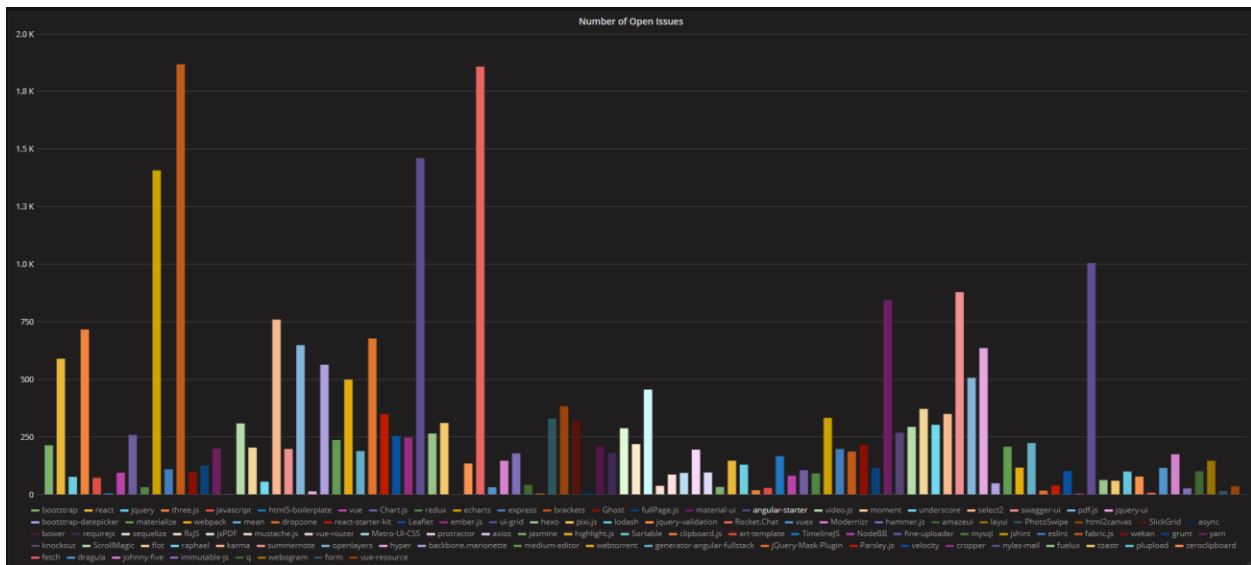


Figure 40: Number of open issues for each project

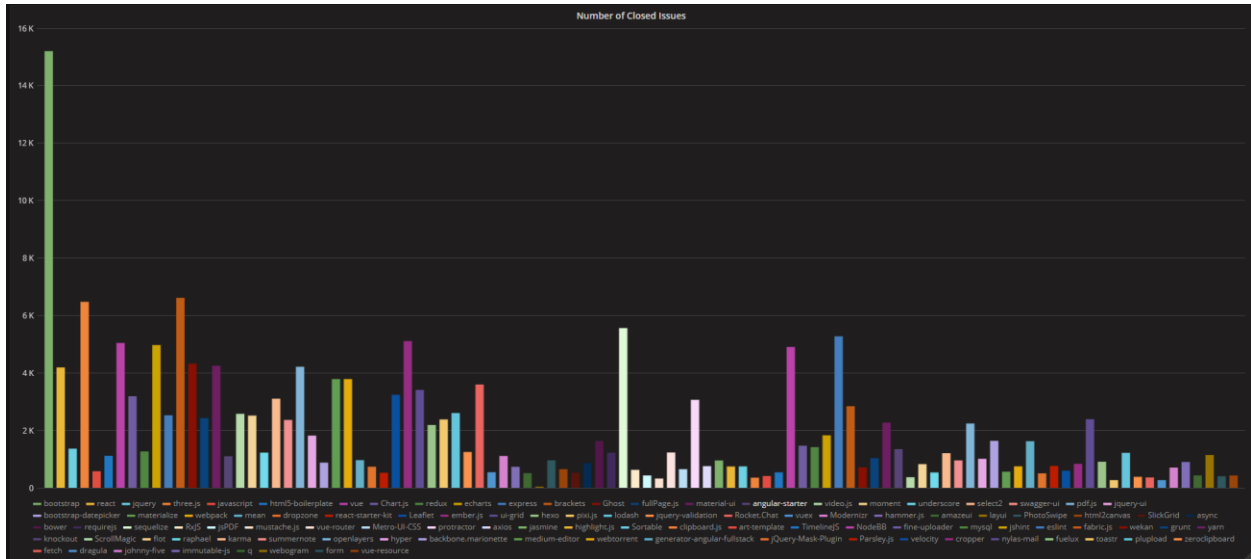


Figure 41: Number of closed issues for each project

## 5.2 Database Results Preparation for Analysis

After we collected data for projects we want to test, in our case written in JavaScript, we export the "releases\_stats" table in a CSV for MS Excel format. The following figures show the exact procedure which is quite simple.

- First, we have to select the exact table we want to export in an MS Excel format.

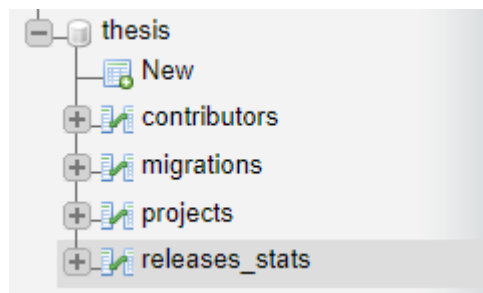


Figure 42: Selection of table

- Then go to Export tab and select as an export method the “Custom – display all possible options” which is the second option of the two. As a format, we choose “CSV for MS Excel” option (Figure 36). Afterwards, we selected the “Put columns names in the first row” option in the “Format-specific options section and pressed the “Go” button to download the file (Figure 37).

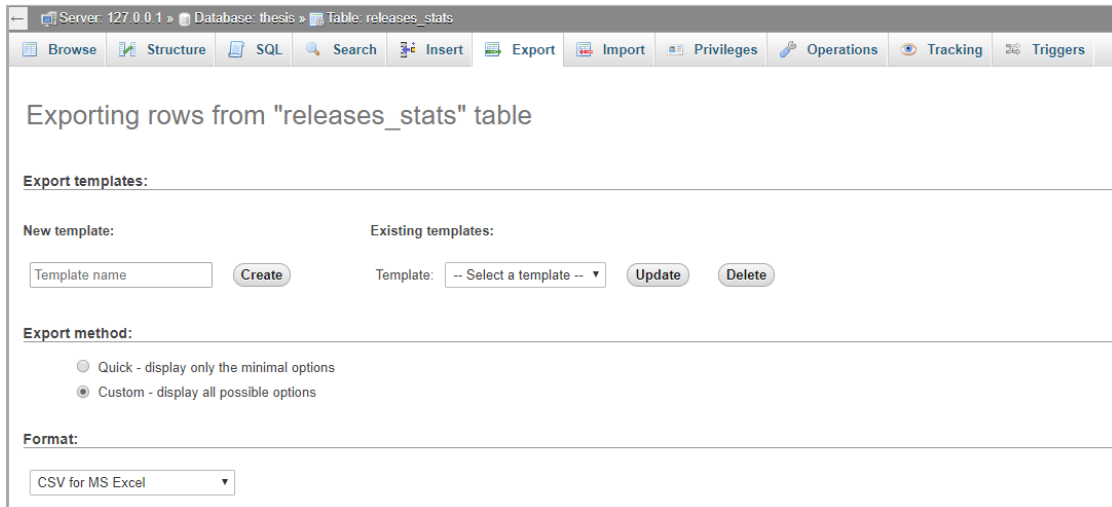


Figure 43: Configurations of the export

**Format-specific options:**

Replace NULL with:

Remove carriage return/line feed characters within columns

Put columns names in the first row

Excel edition:

Figure 44: The last configuration for the export to be performed

After this procedure, we open the file and change “Text to Columns” in “Data” tab of the Excel. By clicking on this option, we gain access to a menu in which we separate the characters by the semicolon (Figure 38, Figure 39). We did all that to move the results from the first cell and make different cells for each one of the metrics. Note that the rel\_date column must be formatted in “Category: Date” and “Type: 3/14/2012”.

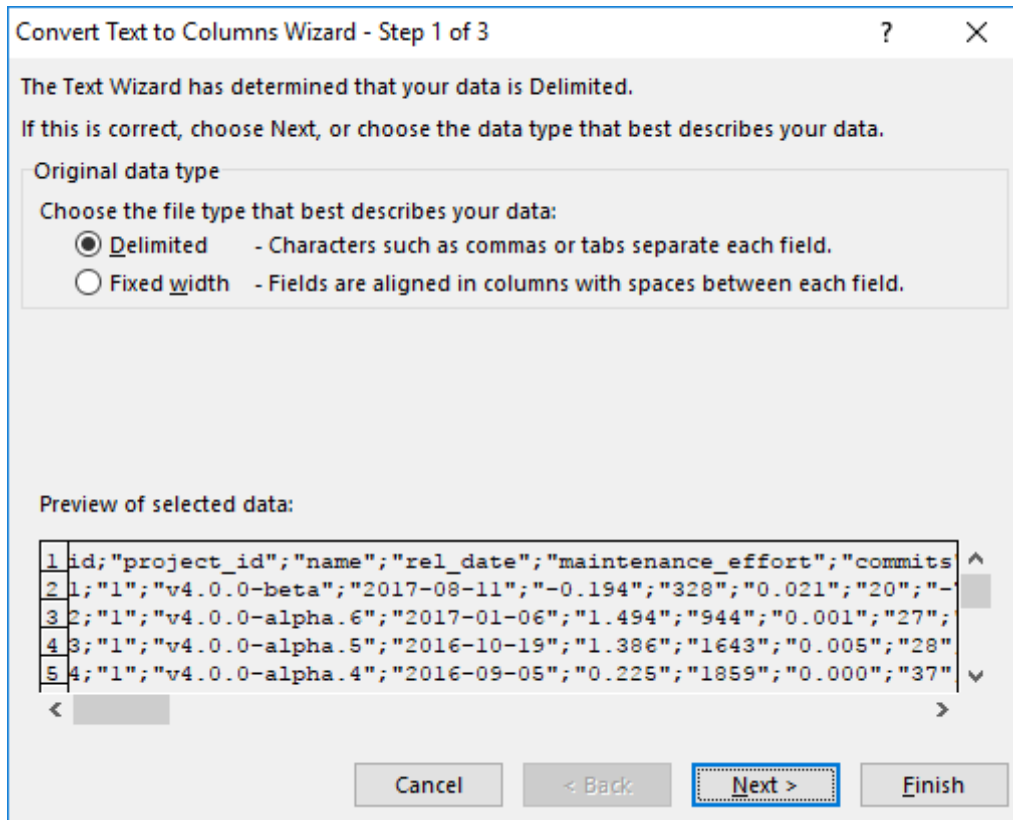


Figure 45: Delimited cells



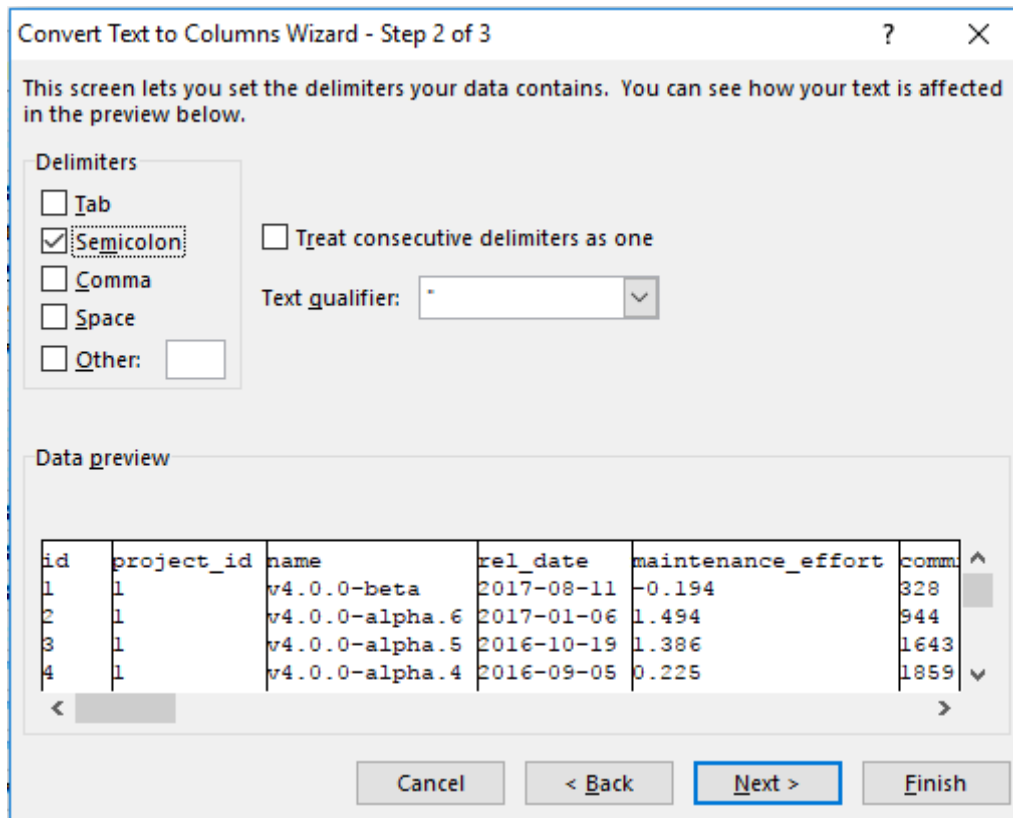


Figure 46: Semicolon option for separation of the entire first column

For the two-sample Kolmogorov-Smirnoff test we need to find the elapsed time in days from the initial release with the following command:

- “=DAYS(all values of rel\_date table for each project\_id, last value of rel\_date table for each project\_id)”

An example of this command for the project\_id=1 would be: “=DAYS( D2, \$D\$43)”. The “\$” symbol is to keep everywhere as a constant the exact cell.

Afterwards, we create a new column that has the theoretical values of growth rate. To do so, we need to execute the command that follows:

- “=IFERROR(POWER(first row in the column we calculated before “:” last row in the column we calculated before,-2/3),0)”

An example of this command for the first 40 projects would be: “=IFERROR(POWER(F2 “:” F3110,-2/3),0)”. The IFERROR() function is set to 0 to protect from division by zero value. Also, POWER() function calculates the theoretical approach of growth rate [64]. So, we power the values in a specific manner which is  $t^{-2/3}$ . The “t” argument is the elapsed time from the initial release we calculated before.

With the use of XLSTAT (trial version) a Mann-Kendall trend test performed for every important metric we calculated. For the last table (Growth Rate) we performed a two-sample Kolmogorov-Smirnoff test. In the section below we check 12 metrics for the whole eight laws. Each metric corresponds to a specific law.

### 5.3 Validation of the Software Evolution

In this chapter, one section for each of the law is going to be presented to validate or not every one of them. We are going to present results and conclusions for 40 projects, but the same attitude can be observed in the other 60 projects. We have acquired measurements for a total of 100 JavaScript projects. For some laws statistically significant conclusions missing, because there aren't enough evidence to declare the law as validated or not. But if there is a lack of a noticeable trend and we are in the previous case, we will announce the law that it might be practically validated or not. Each test has two hypothesis:

- H0: Metric x exhibits no trend
- H1: Metric x exhibits a trend

Where "x" is the exact metric, we tested for each law. The "p-value" is to assure the truth of the result and a "p-value > 0.05" is considered safe to be accepted. The trend for every project generated from the sign of the Slope number. If it is positive, we have a positive trend, and in the other case, it will be opposite. Note that all Grafana charts are presented with the option "stack," to help us make conclusions from these efficiently and accurately. This option sends each project on top of the other and so on.

#### 5.3.1 Law I: Continuing Change

To check the 1<sup>st</sup> law, the "Days Between Releases" (DBR) metric is statistically tested. As we can observe (Table 12) in 2 samples, we haven't any slope which means that there is no evidence of a statistical trend, in 25 we have negative trend and in 13 a positive one. A positive trend in DBR implies that the frequency at which new releases are published decreases. To be more specific a positive trend weakens the validity of the law. However, as we can see we have more projects with a negative trend which corresponds to validation of the law. One important thing is to observe the rate of changes and to do so; we used Grafana to plot a stack (cumulative) of 40 projects (Figure 40). The chart has lots of fluctuations, and an exact rate of change between the commits of new releases cannot be concluded. The percentage of changes seems to increase and the Law to be accurate if we check the number of projects with a positive trend in

comparison to the negative but this hypothesis isn't right for every JavaScript project. In conclusion the **Law I** is **validated**, but the rate of change is unknown.

Days Between Releases (DBR)									
	Project	p-value	Trend	Slope (%)		Project	p-value	Trend	Slope (%)
1	bootstrap	0.004	↓	-1.533	21	swagger-ui	0.043	↑	0.063
2	react	0.626	↑	0.032	22	pdf.js	0.328	↓	-0.200
3	jquery	0.346			23	jquery-ui	0.002	↓	-0.375
4	three.js	< 0.0001	↓	-0.614	24	bootstrap-datepicker	0.036	↓	-0.848
5	javascript	1.000			25	materialize	< 0.0001	↓	-1.267
6	html5-boilerplate	0.003	↓	-2.929	26	webpack	0.001	↓	-0.006
7	vue	0.848			27	mean	0.060	↓	-5.182
8	Chart.js	0.990			28	dropzone	0.001	↓	-0.053
9	redux	~0.000	↓	-0.083	29	react-starter-kit	0.386	↓	-7.050
10	echarts	0.316	↑	0.138	30	Leaflet	0.046	↓	-0.925
11	express	~0.000	↑	0.010	31	ember.js	0.002	↑	0.007
12	brackets	0.132	↑	0.063	32	ui-grid	0.015	↓	-0.063
13	Ghost	0.009	↑	0.063	33	hexo	< 0.0001	↓	-0.087
14	fullPage.js	0.201	↓	-0.099	34	pixi.js	0.202	↓	-0.056
15	material-ui	< 0.0001	↑	0.032	35	lodash	< 0.0001		
16	angular-starter	0.150	↓	-6.0	36	jquery-validation	0.187	↓	-8.706
17	video.js	0.377	↑	0.778	37	Rocket.Chat	0.357		
18	moment	0.133	↓	-0.233	38	vuex	0.05	↓	-0.221
19	underscore	< 0.000	↓	-0.748	39	Modernizr	0.661	↓	-0.3

		1							
20	select2	0.71	↓	-0.168	40	hammer.js	0.139	↓	-0.517

Table 12: Statistical results on law I (continuing change).

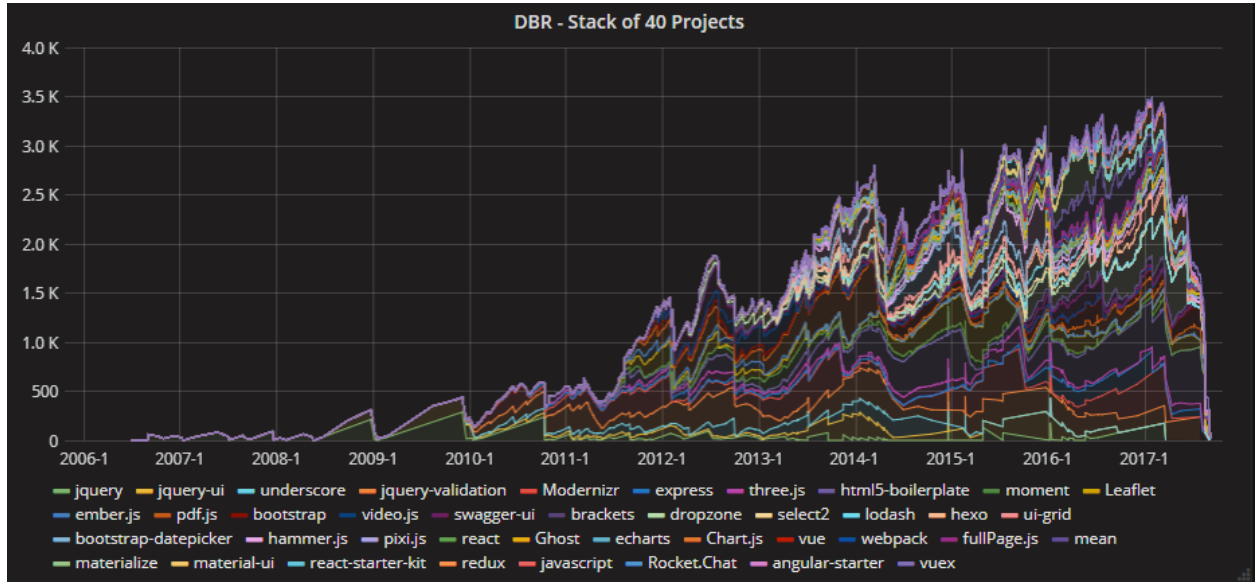


Figure 47: DBR chart for 40 Projects

### 5.3.2 Law II: Increasing complexity

To check the 2<sup>nd</sup> law, the “Complexity” metric is statistically tested. As we can observe (Table 13) in 24 samples, we haven’t any slope which means that there is no evidence of a statistical trend, in 6 we have negative trend and in 10 a positive one. A negative trend or a no trend at all means that Complexity remains at the same levels and weakens the validity of the law. Figure 41 validates the statistical analysis we performed and indicates that there isn’t any slope in most projects. As we can see we have more projects with a negative or no trend which corresponds not to validate the law. Maybe there is a maintenance effort from the developers to keep low the complexity level of the JavaScript software that they produce. In conclusion, the **Law II is not practically and statistically validated.**

Complexity: CCN / LOC									
	Project	p-value	Trend	Slope (%)		Project	p-value	Trend	Slope (%)
1	bootstrap	< 0.0001	↑	0.003	21	swagger-ui	0.004		
2	react	0.000			22	pdf.js	< 0.000	↓	-0.001

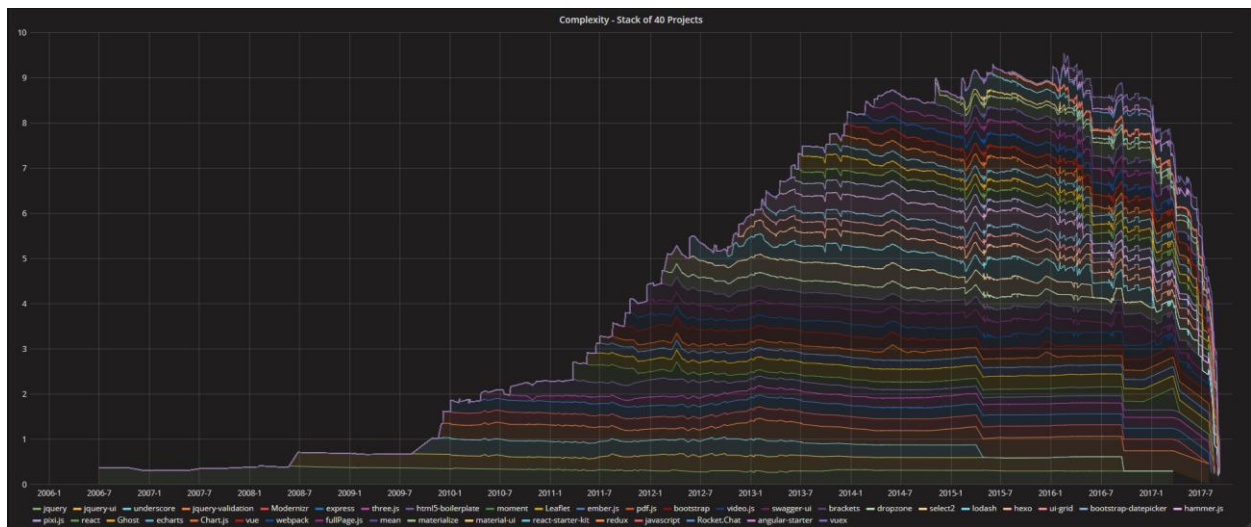
Software Evolution

							1		
3	jquery	< 0.000 1	↑	0.001	23	jquery-ui	0.034		
4	three.js	< 0.000 1	↓	-0.002	24	bootstrap- datepicker	0.048		
5	javascript	< 0.000 1			25	materialize	0.072		
6	html5- boilerplate	0.034	↑	0.006	26	webpack	< 0.000 1		
7	vue	< 0.000 1			27	mean	0.193	↓	-0.001
8	Chart.js	0.002	↑	0.002	28	dropzone	< 0.000 1		
9	redux	< 0.000 1			29	react-starter-kit	0.076	↑	0.026
1 0	echarts	0.008			30	Leaflet	0.008	↓	-0.001
1 1	express	< 0.000 1			31	ember.js	0.621		
1 2	brackets	< 0.000 1	↑	0.001	32	ui-grid	< 0.000 1		
1 3	Ghost	< 0.000 1	↑	0.001	33	hexo	< 0.000 1		
1 4	fullPage.js	< 0.000 1			34	pixi.js	< 0.000 1	↑	0.001
1 5	material-ui	0.048			35	lodash	< 0.000 1		
1 6	angular- starter	0.454			36	jquery-validation	0.029	↓	-0.008
1 7	video.js	0.724			37	Rocket.Chat	< 0.000	↓	-0.001

## Software Evolution

							1		
18	moment	0.174			38	vuex	< 0.0001		
19	underscore	< 0.0001	↑	0.001	39	Modernizr	< 0.0001		
20	select2	0.021	↑	0.001	40	hammer.js	1.000		

*Table 13: Statistical results on law II (increasing complexity).*



*Figure 48: Complexity chart for 40 Projects*

### 5.3.3 Law III: Self-Regulation

To check the 3<sup>rd</sup> law, the “Incremental Growth” metric is statistically tested. As we can observe (Table 14) in 24 samples, we haven’t any slope which means that there is no evidence of a statistical trend, in 6 we have negative trend and in 10 a positive one. A positive trend means that Incremental Growth increases and the validity of the law strengthened. As we can see more than 50% of projects have no trend at all and the other portion remaining has a negative or positive trend. Furthermore, to test the practical validation of the law, we created the Figure 42. In this chart, (Figure 42) there are fluctuations regarding the increase or decrease of incremental growth of methods and functions. If JavaScript projects had more stable gradual growth, we could call the law practically validated, but this doesn’t happen in our tests. In conclusion, the **Law III is practically but not statistically confirmed.**

Incremental Growth									
	Project	p-value	Trend	Slope (%)		Project	p-value	Trend	Slope (%)

Software Evolution

							e		
1	bootstrap	0.777			21	swagger-ui	0.964		
2	react	0.581	↑	0.024	22	pdf.js	0.299	↑	0.198
3	jquery	0.276	↓	-0.243	23	jquery-ui	0.447	↑	0.273
4	three.js	0.002	↑	1.407	24	bootstrap-datepicker	0.910		
5	javascript	0.130			25	materialize	0.187	↑	1.0
6	html5-boilerplate	0.362			26	webpack	0.041		
7	vue	0.482	↑	0.029	27	mean	0.723		
8	Chart.js	0.160	↑	0.406	28	dropzone	0.770		
9	redux	0.561			29	react-starter-kit	0.232	↑	0.367
10	echarts	0.578	↓	-0.337	30	Leaflet	0.427		
11	express	0.642			31	ember.js	0.347		
12	brackets	< 0.0001	↓	-1.691	32	ui-grid	0.481		
13	Ghost	0.208	↓	-0.167	33	hexo	0.094		
14	fullPage.js	0.688			34	pixi.js	0.306		
15	material-ui	0.064	↓	-0.075	35	lodash	0.114		
16	angular-starter	0.036	↑	0.500	36	jquery-validation	0.959		
17	video.js	0.241	↓	-1.0	37	Rocket.Chat	0.793		
18	moment	0.821			38	vuex	0.487		

19	underscore	0.687			39	Modernizr	0.736		
20	select2	0.708	↑	0.033	40	hammer.js	0.367		

Table 14: Statistical results on law III (self-regulation).



Figure 49: Incremental Growth chart for 40 Projects

### 5.3.4 Law IV: Conservation of Organizational Stability

To check the 4<sup>th</sup> law, the “Maintenance Effort” and “Number of Commits” metrics are statistically tested. As we can observe (Table 15) for the “Maintenance Effort” metric in 12 samples, we haven’t any slope which means that there is no evidence of a statistical trend, in 13 we have negative trend and in 15 a positive one. The visual interpretation of Figure 43 indicates that in general, the work rate doesn’t increase or decrease drastically as the projects evolve. Moreover, for the “Number of Commits” variable in 4 samples we haven’t any slope which means that there is no evidence of a statistical trend, in 1 we have negative trend and in 35 a positive one. There is a problem with all these positive trend measurements in “Number of Commits” metric, and it is the p-value which is significantly less than the p-value = 0.05 threshold. To explain it in another way the 35 positive trends are statistically inaccurate. That’s why we plotted the metric with the help of Grafana charts (Figure 44). In Figure 44 we can observe the declining slope indicating that the commits are becoming less as developers publish new releases. The maintenance effort remains the same despite the commits that are reduced over time. In conclusion, the **Law IV is not statistically but practically validated.**

Maintenance Effort									
	Project	p-	Tren	Slope		Project	p-	Tren	Slope



Software Evolution

		value	d	(%)			value	d	(%)
1	bootstrap	0.803	↑	0.008	21	swagger-ui	0.118	↓	-0.032
2	react	0.623			22	pdf.js	0.476	↑	0.016
3	jquery	< 0.000 1	↑	0.009	23	jquery-ui	0.424	↑	0.052
4	three.js	0.014	↑	0.057	24	bootstrap- datepicker	0.549	↓	-0.03
5	javascript	0.224	↑	0.007	25	materialize	0.022	↑	0.309
6	html5- boilerplate	0.718			26	webpack	0.018	↑	0.047
7	vue	0.163			27	mean	1.000	↑	0.009
8	Chart.js	0.958	↓	-0.005	28	dropzone	0.222		
9	redux	0.140	↓	-0.027	29	react-starter-kit	0.158	↑	0.320
10	echarts	0.115	↓	-0.198	30	Leaflet	0.479	↑	0.050
11	express	0.028	↑	0.05	31	ember.js	0.493		
12	brackets	< 0.000 1	↓	-0.252	32	ui-grid	0.944		
13	Ghost	0.007	↓	-0.300	33	hexo	0.266		
14	fullPage.js	0.185	↑	0.021	34	pixi.js	0.455		0.007
15	material-ui	0.401			35	lodash	0.669		
16	angular- starter	0.424	↑	0.090	36	jquery-validation	0.902	↓	-0.003
17	video.js	1.000			37	Rocket.Chat	0.092	↓	-0.053
18	moment	0.307	↓	-0.092	38	vuex	0.562		
19	underscore	0.001	↑	0.149	39	Modernizr	0.404	↓	-0.097
20	select2	0.865	↓	-0.011	40	hammer.js	0.589	↓	-0.044
<b>Number of Commits</b>									
	Project	p- value	Tren d	Slope (%)		Project	p- value	Tren d	Slope (%)
1	bootstrap	<	↑	279.34	21	swagger-ui	<	↑	31.896

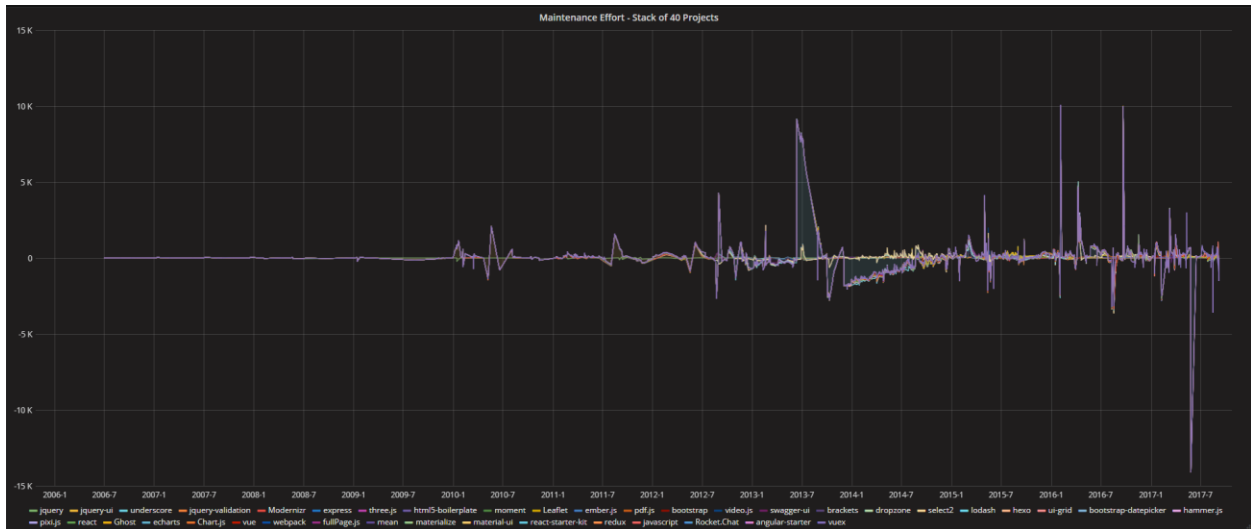
Software Evolution

		0.000 1		8			0.000 1		
2	react	< 0.000 1	↑	121.51 9	22	pdf.js	< 0.000 1	↑	94.610
3	jquery	< 0.000 1	↑	43.019	23	jquery-ui	< 0.000 1	↑	4.775
4	three.js	0.057	↑	1.396	24	bootstrap- datepicker	< 0.000 1	↑	39.929
5	javascript	< 0.000 1	↑	11.6	25	materialize	< 0.000 1	↑	87.158
6	html5- boilerplate	< 0.000 1	↑	50.0	26	webpack	< 0.000 1	↑	8.029
7	vue	< 0.000 1			27	mean	< 0.000 1	↑	96.857
8	Chart.js	0.564	↓	-0.154	28	dropzone	< 0.000 1	↑	8.861
9	redux	< 0.000 1	↑	49.339	29	react-starter-kit	0.001	↑	57.833
1 0	echarts	< 0.000 1	↑	71.308	30	Leaflet	0.001	↑	142.80 6
1 1	express	< 0.000 1	↑	12.548	31	ember.js	< 0.000 1	↑	44.091
1 2	brackets	< 0.000 1	↑	123.94 7	32	ui-grid	< 0.000 1	↑	53.960
1 3	Ghost	< 0.000 1	↑	64.558	33	hexo	0.730		
1 4	fullPage.js	< 0.000 1	↑	12.890	34	pixi.js	< 0.000 1	↑	63.000
1 5	material-ui	0.790			35	lodash	< 0.000		

## Software Evolution

							1		
16	angular-starter	< 0.0001	↑	69.667	36	jquery-validation	< 0.0001	↑	51.718
17	video.js	0.015	↑	4.2	37	Rocket.Chat	< 0.0001	↑	90.975
18	moment	< 0.0001	↑	64.929	38	vuex	< 0.0001	↑	5.459
19	underscore	< 0.0001	↑	19.593	39	Modernizr	< 0.0001	↑	78.0
20	select2	< 0.0001	↑	64.333	40	hammer.js	< 0.0001	↑	42.415

*Table 15: Statistical results on law IV (conservation of organizational stability).*



*Figure 50: Maintenance Effort chart for 40 Projects*

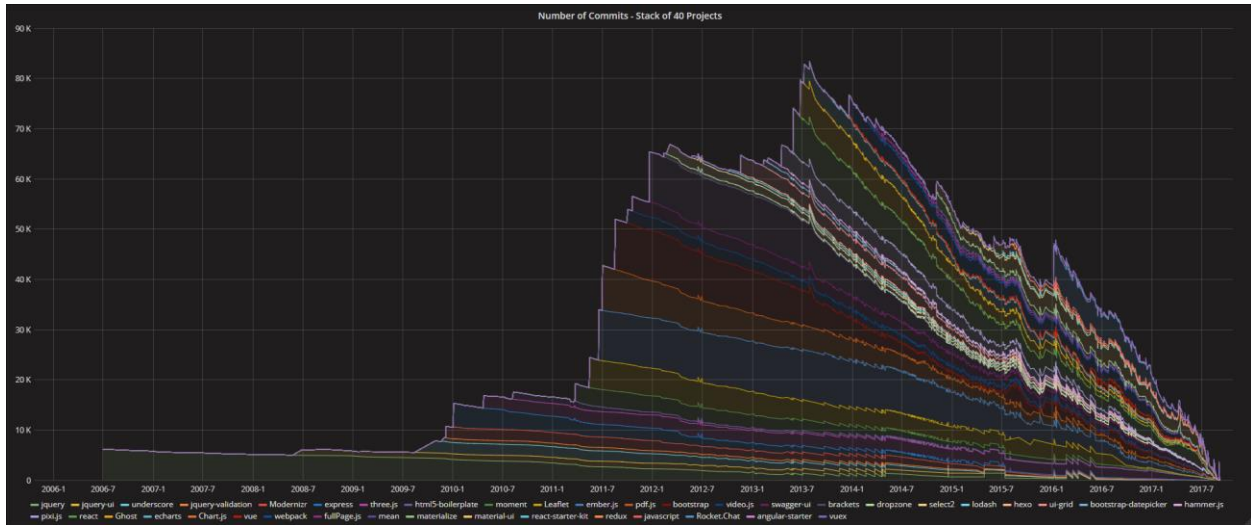


Figure 51: Number of Commits chart for 40 Projects

### 5.3.5 Law V: Conservation of Familiarity

To check the 5<sup>th</sup> law, the “Incremental Changes” metric is statistically tested. As we can observe (Table 16) in 4 samples, we haven’t any slope which means that there is no evidence of a statistical trend, in 19 we have negative trend and in 17 a positive one. Figure 45 has identical results as “Incremental Growth” of the 3<sup>rd</sup> law with the only difference that the graph is positively displaced because of the addition of changed functions and methods. We combined the statistical and practical results and observed that in some projects the trend is increasing and in others decreasing but not throughout the whole project's releases. These fluctuations that are presented and the divided in half statistical results conclude that some projects have a positive trend which implies that the number of functions is added or changed increases, but it will be wrong to happen continuously in every project. On the other hand, other projects have a negative trend which implies that fewer functions are added or changed over time, and the project is almost extinct. In conclusion, the **Law V is not practically and statistically validated.**

Incremental Changes									
	Project	p-value	Trend	Slope (%)		Project	p-value	Trend	Slope (%)
1	bootstrap	0.259	↓	-0.7	21	swagger-ui	0.497	↑	0.029
2	react	0.101	↓	-0.514	22	pdf.js	0.571	↑	0.629
3	jquery	0.096	↓	-0.515	23	jquery-ui	0.666	↑	0.412
4	three.js	0.006	↑	1.4	24	bootstrap-datepicker	0.398	↓	-0.699
5	javascript	~0.000	↑	0.133	25	materialize	0.205	↑	1.7

## Software Evolution

6	html5-boilerplate	0.895			26	webpack	0.041		
7	vue	0.577	↑	0.056	27	mean	0.487	↓	-3.1
8	Chart.js	0.340	↑	0.442	28	dropzone	0.777		
9	redux	0.439	↑	0.220	29	react-starter-kit	0.902	↑	0.458
10	echarts	0.293	↓	-1.427	30	Leaflet	0.320	↑	0.628
11	express	0.485	↑	0.065	31	ember.js	0.381	↓	-0.033
12	brackets	< 0.0001	↓	-3.381	32	ui-grid	0.549	↑	0.400
13	Ghost	0.111	↓	-0.286	33	hexo	0.117	↓	-0.263
14	fullPage.js	0.373	↑	0.067	34	pixi.js	0.086	↑	0.212
15	material-ui	0.041	↓	-0.164	35	lodash	0.114		
16	angular-starter	0.085	↑	1.200	36	jquery-validation	0.650	↓	-0.833
17	video.js	0.390	↓	-3.000	37	Rocket.Chat	0.775	↓	-0.045
18	moment	0.752	↓	-0.278	38	vuex	0.710	↓	-0.131
19	underscore	0.425	↑	0.057	39	Modernizr	0.786	↓	-0.053
20	select2	0.889	↓	-0.270	40	hammer.js	0.052	↓	-1.928

*Table 16: Statistical results on law V (conservation of familiarity).*



*Figure 52: Incremental Changes chart for 40 Projects*

### 5.3.6 Law VI: Continuing Growth

To check the 6<sup>th</sup> law, the “Lines of Code” metric is statistically tested. As we can observe (Table 17) in 1 samples, we haven’t any slope which means that there is no evidence of a statistical

trend, in 34 we have negative trend and in 2 a positive one. Despite the biggest amount of negative trend, in general, the p-value is far less than the accepted 0.05 value. However, the two projects with positive trend have a p-value > 0.05. Figure 46 validates the statistical analysis and reinforces the results. There is an increasing trend for the JavaScript projects, and programmers keep adding new code to enhance the offered functionality. In addition, one important thing we noticed is the last part of Figure 46 which points out that developers try to reduce the Lines of Code (LOC) and improve the software with only the necessary lines for functionalities they provide. This effort has been increased in our contemporary years and at the latest versions. In conclusion, the **Law VI** is **practically and statistically validated**.

Lines of Code (LOC)									
	Project	p-value	Trend	Slope (%)		Project	p-value	Trend	Slope (%)
1	bootstrap	~0.000	↓	-343	21	swagger-ui	N/A	N/A	N/A
2	react	< 0.0001	↓	-1172	22	pdf.js	N/A	N/A	N/A
3	jquery	< 0.0001	↓	-308.89	23	jquery-ui	< 0.0001	↓	-148.45
4	three.js	< 0.0001	↓	-2162.5	24	bootstrap-datepicker	0.360	↓	-16.043
5	javascript	< 0.0001	↓	-9.6	25	materialize	< 0.0001	↓	-322.92
6	html5-boilerplate	0.006	↑	215.525	26	webpack	< 0.0001	↓	-3247.1
7	vue	< 0.0001		-207.66	27	mean	< 0.0001	↓	-121.37
8	Chart.js	< 0.0001	↓	-1026.7	28	dropzone	0.040	↓	-83.703
9	redux	< 0.0001	↓	-85.810	29	react-starter-kit	< 0.0001	↓	-66.453
10	echarts	0.287	↓	-288.35	30	Leaflet	< 0.0001	↓	-416.66
11	express	N/A	N/A	N/A	31	ember.js	< 0.0001	↓	-64.9
12	brackets	< 0.0001	↓	-19.000	32	ui-grid	< 0.0001	↓	-468.03
13	Ghost	< 0.0001	↓	-1969.9	33	hexo	< 0.0001	↓	-57.364
14	fullPage.js	< 0.0001	↓	-318.51	34	pixi.js	< 0.0001	↓	-78.152
15	material-ui	< 0.0001	↓	-127.39	35	lodash	0.002	↓	-125.29

16	angular-starter	< 0.0001	↓	-168.11	36	jquery-validation	< 0.0001	↓	-222.23
17	video.js	0.284	↓	-1.0	37	Rocket.Chat	0.352	↑	16.995
18	moment	0.010	↓	-28.2	38	vuex	0.016	↓	-1248.6
19	underscore	< 0.0001	↓	-2357.3	39	Modernizr	< 0.0001	↓	-56.471
20	select2	< 0.0001	↓	-68.069	40	hammer.js	< 0.0001	↓	-52.334

Table 17: Statistical results on law VI (continuing growth).

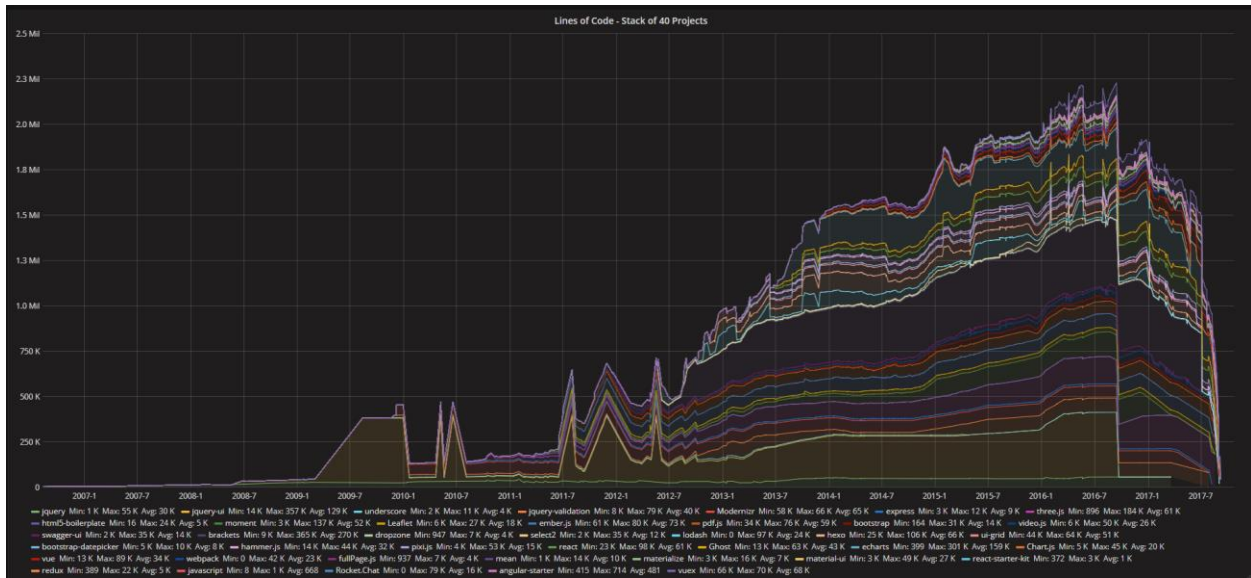


Figure 53: Lines of Code chart for 40 Projects

### 5.3.7 Law VII: Declining Quality

To check the 7<sup>th</sup> law, the “Depth of Inheritance Tree,” “Comment Rate,” “Maintainability,” and “Number of bug-related commits” metrics are statistically tested. As we can observe (Table 18) for “DIT” variable in 18 samples, we haven’t any slope which means that there is no evidence of a statistical trend, in 2 we have a negative trend. The p-value is low, and we can’t reach to a result from the DIT value, only that it remains at equal levels if we examine the entire range of the projects. For “Comment Rate” (CR) variable in 8 samples we haven’t any slope which means that there is no evidence of a statistical trend, in 13 we have negative trend and in 19 a positive one. The “Comment Rate” shows us that it has an attitude to increase. For “Maintainability” variable in 17 samples, we haven’t any slope which means that there is no evidence of a statistical trend, in 7 we have negative trend and in 15 a positive one. The “Maintainability” shows us that in general is not increasing or decreasing, but in some cases, it has an attitude to

increase. An increase in “Maintainability” and “Comment Rate” are an indication of improvement in the quality. Furthermore, for “Number of bug-related commits” variable in 40 samples we haven’t any slope which means that there is no evidence of a statistical trend. By examining each project separately, we can identify a general increase in “Maintainability” and “CR” metrics of and at least a neutral state for the other two remaining. In other words, the quality is at equal levels or maybe increases a bit. In conclusion, the **Law VII** is **not statistically validated**.

Depth of Inheritance Tree (DIT)									
	Project	p-value	Trend	Slope (%)		Project	p-value	Trend	Slope (%)
1	bootstrap	< 0.0001			21	swagger-ui	N/A	N/A	N/A
2	react	< 0.0001			22	pdf.js	N/A	N/A	N/A
3	jquery	N/A	N/A	N/A	23	jquery-ui	N/A	N/A	N/A
4	three.js	0.091			24	bootstrap-datepicker	N/A	N/A	N/A
5	javascript	N/A	N/A	N/A	25	materialize	0.098		
6	html5-boilerplate	N/A	N/A	N/A	26	webpack	0.001		
7	vue	N/A	N/A	N/A	27	mean	0.021		
8	Chart.js	< 0.0001			28	dropzone	N/A	N/A	N/A
9	redux	0.096			29	react-starter-kit	0.015		
10	echarts	0.018			30	Leaflet	0.012		
11	express	< 0.0001			31	ember.js	N/A	N/A	N/A
12	brackets	0.090			32	ui-grid	N/A	N/A	N/A
13	Ghost	< 0.0001			33	hexo	< 0.0001	↓	-0.071
14	fullPage.js	0.018			34	pixi.js	< 0.0001		
15	material-ui	N/A	N/A	N/A	35	lodash	N/A	N/A	N/A
16	angular-starter	N/A	N/A	N/A	36	jquery-validation	N/A	N/A	N/A
17	video.js	0.045			37	Rocket.Chat	N/A	N/A	N/A
18	moment	< 0.0001			38	vuex	N/A	N/A	N/A



Software Evolution

19	underscore	N/A	N/A	N/A	39	Modernizr	N/A	N/A	N/A
20	select2	< 0.0001	↓	-0.161	40	hammer.js	N/A	N/A	N/A
<b>Comment Rate (CR)</b>									
	Project	p-value	Trend	Slope (%)		Project	p-value	Trend	Slope (%)
1	bootstrap	0.003	↑	0.043	21	swagger-ui	< 0.0001	↑	0.08
2	react	< 0.0001	↑	0.122	22	pdf.js	0.513		
3	jquery	0.024	↓	-0.021	23	jquery-ui	0.000	↓	-0.055
4	three.js	< 0.0001	↓	-0.091	24	bootstrap-datepicker	0.001	↑	0.026
5	javascript	< 0.0001	↓	-0.065	25	materialize	0.016	↑	0.080
6	html5-boilerplate	0.385	↓	-0.073	26	webpack	< 0.0001	↑	0.009
7	vue	< 0.0001	↑	0.047	27	mean	0.027	↑	0.270
8	Chart.js	0.012	↑	0.075	28	dropzone	< 0.0001	↑	0.114
9	redux	0.121	↑	0.012	29	react-starter-kit	0.003	↑	1.708
10	echarts	0.003	↓	-0.049	30	Leaflet	0.622		
11	express	< 0.0001	↑	0.015	31	ember.js	< 0.0001	↑	0.015
12	brackets	< 0.0001	↓	-0.016	32	ui-grid	0.639		
13	Ghost	< 0.0001	↑	0.043	33	hexo	< 0.0001	↓	-0.039
14	fullPage.js	0.531			34	pixi.js	0.570		
15	material-ui	< 0.0001	↓	-0.009	35	lodash	< 0.0001	↑	0.142
16	angular-starter	0.564			36	jquery-validation	0.152	↓	-0.036
17	video.js	0.186	↑	0.017	37	Rocket.Chat	0.053	↓	-0.019
18	moment	0.001	↑	0.068	38	vuex	< 0.0001	↓	-0.005
19	underscore	< 0.0001	↑	0.061	39	Modernizr	0.004		0.013
20	select2	0.602			40	hammer.js	< 0.0001	↓	-0.160

Software Evolution

<b>Maintainability</b>									
	Project	p-value	Trend	Slope (%)		Project	p-value	Trend	Slope (%)
1	bootstrap	~0.000	↑	0.033	21	swagger-ui	0.081	↑	0.004
2	react	0.623			22	pdf.js	0.368		
3	jquery	< 0.0001	↑	0.009	23	jquery-ui	0.028	↑	0.002
4	three.js	< 0.0001	↓	-0.017	24	bootstrap-datepicker	0.160	↑	0.005
5	javascript	0.001			25	materialize	0.168	↑	0.003
6	html5-boilerplate	0.001	↑	0.11	26	webpack	< 0.0001	↑	0.004
7	vue	< 0.0001	↓	-0.002	27	mean	0.674		
8	Chart.js	< 0.0001	↑	0.011	28	dropzone	0.222		
9	redux	0.071			29	react-starter-kit	0.158	↑	0.32
10	echarts	0.571			30	Leaflet	0.015	↓	-0.006
11	express	~0.000			31	ember.js	< 0.0001	↓	-0.001
12	brackets	< 0.0001	↑	0.009	32	ui-grid	< 0.0001		
13	Ghost	0.118			33	hexo	~0.000	↓	-0.002
14	fullPage.js	0.249			34	pixi.js	< 0.0001	↑	0.034
15	material-ui	< 0.0001	↑	0.002	35	lodash	< 0.0001	↑	0.004
16	angular-starter	0.648			36	jquery-validation	0.011	↓	-0.213
17	video.js	0.724			37	Rocket.Chat	0.293		
18	moment	0.252	↓	-0.007	38	vuex	N/A	N/A	N/A
19	underscore	0.583			39	Modernizr	< 0.0001		
20	select2	0.196	↑	0.007	40	hammer.js	0.807		
<b>Number of bug-related commits</b>									
	Project	p-value	Trend	Slope (%)		Project	p-value	Trend	Slope (%)
1	bootstrap	0.001			21	swagger-ui	0.990		

2	react	0.020			22	pdf.js	0.255		
3	jquery	< 0.0001			23	jquery-ui	0.926		
4	three.js	0.001			24	bootstrap- datepicker	0.914		
5	javascript				25	materialize	0.957		
6	html5- boilerplate	0.270			26	webpack	< 0.0001		
7	vue	0.338			27	mean	0.391		
8	Chart.js	0.412			28	dropzone	0.314		
9	redux	0.891			29	react-starter-kit	0.511		
10	echarts	0.053			30	Leaflet	0.940		
11	express	0.001			31	ember.js	0.094		
12	brackets	0.017			32	ui-grid	0.261		
13	Ghost	0.106			33	hexo	0.028		
14	fullPage.js	0.163			34	pixi.js	0.087		
15	material-ui	0.088			35	lodash	~0.000		
16	angular- starter	0.429			36	jquery-validation	0.453		
17	video.js	0.349			37	Rocket.Chat	0.293		
18	moment	0.621			38	vuex	0.039		
19	underscore	0.059			39	Modernizr	0.919		
20	select2	0.034			40	hammer.js	0.271		

Table 18: Statistical results on law VII (declining quality).

### 5.3.8 Law VIII: Feedback System

To check the 8<sup>th</sup> law, a statistical comparison between “Growth Rate” and Theoretical Growth Rate was performed. Theoretical Growth Rate is defined as  $t^{-2/3}$  where “t” is the elapsed time in days from the initial release [64]. We executed a two-sample Kolmogorov-Smirnoff test [65] and compared the actual evolution with the theoretical. As we can observe (Table 19) in 40 samples, the p-value is less than 0.0001. We didn’t have any value matching between the two variables tested in all the projects. In conclusion, the **Law VI is not statistically validated**.

Growth Rate					
	Project	p-value		Project	p-value
1	bootstrap	0.000	21	swagger-ui	< 0.0001

2	react	< 0.0001	22	pdf.js	< 0.0001
3	jquery	< 0.0001	23	jquery-ui	< 0.0001
4	three.js	< 0.0001	24	bootstrap-datepicker	< 0.0001
5	javascript	< 0.0001	25	materialize	< 0.0001
6	html5-boilerplate	< 0.0001	26	webpack	< 0.0001
7	vue	< 0.0001	27	mean	< 0.0001
8	Chart.js	< 0.0001	28	dropzone	< 0.0001
9	redux	< 0.0001	29	react-starter-kit	< 0.0001
10	echarts	< 0.0001	30	Leaflet	< 0.0001
11	express	< 0.0001	31	ember.js	< 0.0001
12	brackets	< 0.0001	32	ui-grid	< 0.0001
13	Ghost	< 0.0001	33	hexo	< 0.0001
14	fullPage.js	< 0.0001	34	pixi.js	< 0.0001
15	material-ui	< 0.0001	35	lodash	< 0.0001
16	angular-starter	< 0.0001	36	jquery-validation	< 0.0001
17	video.js	< 0.0001	37	Rocket.Chat	< 0.0001
18	moment	< 0.0001	38	vuex	< 0.0001
19	underscore	< 0.0001	39	Modernizr	< 0.0001
20	select2	< 0.0001	40	hammer.js	< 0.0001

Table 19: Statistical results on law VIII (feedback system).

## 5.4 Conclusions and Comparison to other studies

In chapter 2.1.2 we have introduced a table with results from other studies. Now that we obtained results, we are going to conclude and compare them to various studies. The summary table follows for our results (Table 20):

Law - Hypothesis	Our Finding for JavaScript Projects
I – System continuously change	True.
II – Complexity rises	False. Complexity remains almost the same.
III – Incremental growth exhibits adjustments	True. Practically validated.
IV – Work rate is constant	True. Practically validated.
V – Incremental changes remain invariant	False. Lots of fluctuations.
VI – Continuously grow	True.
VII – Quality declines	False. Quality remains almost the same or in some cases increases.
VIII – Growth rate drops at a rate comparable to $t^{-2/3}$	False. The growth rate doesn't decrease that rapidly.

Table 20: Validation of the laws

We added our study in the table of the chapter 2.1.2 to summarize our results and conclusions (Table 21):

Reference	Year	Programming Language	Number of Projects	I	II	III	IV	V	VI	VII	VIII
Godfrey & Tu [52, 53]	2000 and 2001	C	5	Y			N		Y		N
Robles et al. [54]	2005	C, C++, Java	19	Y			N		Y		N
Mens et al. [55]	2008	Java	1	Y	N				Y		
Xie et al. [56]	2009	C	7	Y	Y	Y		N	Y	N	N
Israeli & Feiteison [57]	2010	C	1	Y	N	Y	Y		Y	N	Y
Businge et al. [58]	2010	Java	21	Y		Y		N	Y		
Neamtii et al. [59]	2013	C	9	Y	N	N	N	N	Y	N	N
Kaur et al. [60]	2014	C++	2	Y	Y	Y		Y	Y	Y	
Amanatidis & Chatzigeorgiou 2015[13]	2015	PHP	30	Y	N	Y	Y	Y	Y		N
<b>This study</b>	<b>2017</b>	<b>JavaScript</b>	<b>100</b>	<b>Y</b>	<b>N</b>	<b>Y</b>	<b>Y</b>	<b>N</b>	<b>Y</b>	<b>N</b>	<b>N</b>

Table 21: Studies about the validity of Lehman's laws including ours

If we compare our results to the others we conclude to the following observations:

- The 1<sup>st</sup> law is validated to all programming languages.
- The 2<sup>nd</sup> law is not validated in JavaScript, PHP, and Java. C and C++ have different outcomes regarding the specific study we notice.
- The 3<sup>rd</sup> law is validated in JavaScript, PHP, and C++. C and Java have different outcomes regarding the specific study we notice.
- The 4<sup>th</sup> law is validated in JavaScript and PHP. C, C, and C++ projects don't verify the law. The exception is one study related to C.
- The 5<sup>th</sup> law is validated in C++ and PHP. JavaScript, C, and Java don't confirm the law.
- The 6<sup>th</sup> law is validated to all programming languages.
- The 7<sup>th</sup> law is validated from only one study about C++ programming languages. Our thesis and the others don't support the law.

- The 8<sup>th</sup> law is validated only by one study for projects written in C programming language. All the others don't endorse the 8<sup>th</sup> law.

## Bibliographic References

- [1] Michael W. Godfrey and Daniel M. German "On the Evolution of Lehman's Laws" JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS J. Softw. Evol. and Proc. 0000; 00:1–7 Published online in Wiley InterScience ([www.interscience.wiley.com](http://www.interscience.wiley.com)). DOI: 10.1002/smr.
- [2] Lehman, M. M. (1980). "On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle." Journal of Systems and Software. 1: 213–221. doi: 10.1016/0164-1212(79)90022-0.
- [3] Liguu Yu and Alok Mishra (2013) "An Empirical Study of Lehman's Law on Software Quality Evolution in International" Journal of Software and Informatics, 11/2013; 7(3):469-481.
- [4] Press release announcing JavaScript, "Netscape and Sun announce JavaScript," PR Newswire, December 4, 1995
- [5] "Standard ECMA-262". Ecma International. 2017-07-03.
- [6] "ECMAScript Language Overview" (PDF). 2007-10-23. p. 4. Retrieved 2009-05-03.
- [7] Williams, Alex (9 July 2012). "GitHub Pours Energies into Enterprise – Raises \$100 Million From Power VC Andreessen Horowitz". TechCrunch. "Andreessen Horowitz is investing an eye-popping \$100 million into GitHub".
- [8] "Why GitHub's pricing model stinks (for us)." LosTechies. 7 November 2012. Archived from the original on 29 June 2015. Retrieved 29 June 2015.
- [9] "The Problem With Putting All the World's Code in GitHub." Wired. 29 June 2015. Archived from the original on 29 June 2015. Retrieved 29 June 2015.
- [10] "Celebrating nine years of GitHub with an anniversary sale." github.com. Github. Retrieved 2017-04-11.
- [11] Gousios, Georgios; Vasilescu, Bogdan; Serebrenik, Alexander; Zaidman, Andy. "Lean GHTorrent: GitHub Data on Demand" (PDF). The Netherlands: Delft University of Technology & †Eindhoven University of Technology: 1. Retrieved 9 July 2014. During recent years, GITHUB (2008) has become the largest code host in the world.
- [12] Iulian Neamtiu, Guowu Xie and Jianbo Chen "Towards a better understanding of software evolution: an empirical study on open-source software" JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS J. Softw.: Evol. and Proc. 2013; 25:193–218 Published online 1 September 2011 in Wiley Online Library ([www.wileyonlinelibrary.com](http://www.wileyonlinelibrary.com)). DOI: 10.1002/smr.564.

- [13] Theodoros Amanatidis, Alexander Chatzigeorgiou "Studying the evolution of PHP web applications" *Information and Software Technology* 72 (2016) 48–67 Contents lists available at Science Direct *Information and Software Technology* journal homepage: [www.elsevier.com/locate/infsof](http://www.elsevier.com/locate/infsof).
- [14] Flanagan 2006, p. 16.
- [15] "JavaScript data types and data structures - JavaScript | MDN". *Developer.mozilla.org*. 2017-02 16. Retrieved 2017-02-24.
- [16] "Inheritance and the prototype chain." *Mozilla Developer Network*. Mozilla. Retrieved 6 April 2013.
- [17] Haverbeke, Marijn (2011). "Eloquent JavaScript." No Starch Press. pp. 95–97. ISBN 978-1-59327-282-1.
- [18] "Properties of the Function Object." *Es5.github.com*. Retrieved 2013-05-26.
- [19] Flanagan 2006, p. 141.
- [20] "[Concurrency model and Event Loop](#)". *Mozilla Developer Network*. Retrieved 2015-08-28.
- [21] Haverbeke, Marijn (2011). "Eloquent JavaScript." No Starch Press. pp. 139–149. ISBN 978-1-59327-282-1.
- [22] John Resig, "[JavaScript Getters and Setters](#)," *Ejohn.org*, 18 July 2007, accessed 2 January 2010.
- [23] "[E4X – Archive of obsolete content | MDN](#)". *Mozilla Developer Network*. Mozilla Foundation. Feb 14, 2014. Retrieved 13 July 2014.
- [24] Mahemoff, Michael (17 December 2009). "[Server-Side JavaScript, Back with a Vengeance](#)". *readwrite.com*. Retrieved 2016-07-16.
- [25] Crockford, D (2001) *JavaScript: The World's Most Misunderstood Programming Language* [Online]. Available from <http://www.crockford.com/javascript/javascript.html> (Accessed: 10 October 2010).
- [26] The PHP Group (2010) *History of PHP* [Online]. Available from <http://www.php.net/manual/en/history.php.php> (Accessed: 10 October 2010).
- [27] RASPBIAN STRETCH WITH DESKTOP. <https://www.raspberrypi.org/downloads/raspbian/>. August 2017.
- [28] "Interview with Kai Seidler from the XAMPP project." *MySQL AB*. Retrieved 2015-06-07.
- [29] Daniel Gafitescu (June 6, 2013). "Goodbye CodeIgniter, Hello Laravel." [www.sitepoint.com](http://www.sitepoint.com). Retrieved December 21, 2013.

- [30] Martin Bean (April 2015). Laravel 5 Essentials. [www.books.google.com](http://www.books.google.com). Packt. ISBN 978-1785283017. Retrieved September 2, 2015.
- [31] "What is MySQL?" MySQL 5.1 Reference Manual. Oracle. Retrieved 17 September 2012. The official way to pronounce "MySQL" is "My Ess Que Ell" (not "my sequel").
- [32] "Goutte, a simple PHP Web Scraper." <https://github.com/FriendsOfPHP/Goutte>. Last version v.3.2.1. 03 January 2017.
- [33] Scopatz, Anthony; Huff, Kathryn D. (2015). Effective Computation in Physics. O'Reilly Media, Inc. p. 351. ISBN 9781491901595. Retrieved 20 April 2016.
- [34] Torvalds, Linus (2005-04-07). "Re: Kernel SCM saga..". Linux-kernel (Mailing list). "So I'm writing some scripts to try to track things a whole lot faster."
- [35] Torvalds, Linus (2007-06-10). "Re: fatal: serious inflate inconsistency". git (Mailing list).
- [36] Linus Torvalds (2007-05-03). Google tech talk: Linus Torvalds on git. Event occurs at 02:30. Retrieved 2007-05-16.
- [37] Freddy Mallet (20 March 2013). "SONAR is becoming SONARQUBE". SonarQube project mailing list. Retrieved 3 July 2013.
- [38] Mariano (2009-11-17). "Creating a Sonar Plugin for software development metrics". Archived from the original on March 24, 2010. Retrieved 2017-08-29.
- [39] Hazrati, Vikas (2010-03-30). "Monetizing the Technical Debt". Retrieved 2017-08-29.
- [40] "Methods and Tools issue"(PDF). 2010-03-01. Retrieved 2017-08-29.
- [41] Campell/Papapetrou, Ann/Patroklos (2013). Sonar (SonarQube) in action. Greenwich, Connecticut, USA: Manning Publications. p. 350. ISBN 978-1617290954.
- [42] Leonardo Humberto Silva, Daniel Hovadick, Marco Tulio Valente, Alexandre Bergel, Nicolas Anquetil, Anne Etien "JSClassFinder: A Tool to Detect Class-like Structures in JavaScript" arXiv:1602.05891v1 [cs.SE] 18 Feb 2016.
- [43] <https://grafana.com/>. 2017.
- [44] <https://www.apachefriends.org/download.html>. 2017.
- [45] <https://www.sonarqube.org/downloads/>. SonarQube 6.5. Aug 3, 2017.
- [46] <https://git-scm.com/downloads>. Latest source Release: 2.14.1. 2017-08-04.
- [47] <https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner>. 2017.
- [48] <https://autohotkey.com/download/>. v1.1.26.01 - July 16, 2017.



- [49] <https://github.com/aserg-ufmg/JSClassFinder>. 2017.
- [50] <https://grafana.com/grafana/download?platform=windows>. 2017.
- [51] /Lehman, 1985a/. "Software Evolution - Processes of Software Change". London 1985.
- [52] M.W. Godfrey, Q.Tu. Evolution in open source software: a case study, in Proceedings of the International Conference on Software Maintenance (ICSM'00), Washington, DC, USA, 2000, p.131.
- [53] M. Godfrey, Q. Tu, Growth, evolution, and structural change in open source software, in Proceedings of the 4<sup>th</sup> International Workshop On Principles Of Software Evolution, New York, NY, USA, pp. 103-106.
- [54] G. Robles, J.J. Amor, J.M. Gonzalez-Barahona, I. Herraiz, Evolution and growth in large libre software projects, in Proceedings of Eight International Workshop on Principles of Software Evolution, 2005, pp. 165-174.
- [55] T. Mens, J. Fernandez-Ramil, S. Degrandart, The evolution of Eclipse, in Proceedings of IEEE International Conference on Software Maintenance, 2008. ICSM2008, 2008, pp.386–395.
- [56] G. Xie, J. Chen, I. Neamtii, Towards a better understanding of software evolution: an empirical study on open source software, in Proceedings of IEEE International Conference on Software Maintenance, ICSM2009, 2009, pp.51–60.
- [57] A. Israeli, D.G. Feitelson. The Linux kernel as a case study in software evolution, J. Syst. Softw. 83 (3) (Mar.2010) 485–501.
- [58] J. Businge, A. Serebrenik, M. van den Brand, An empirical study of the evolution of eclipse third-party plug-ins, in: Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), New York, NY, USA, 2010, pp.63–72.
- [59] I. Neamtii, G. Xie, J. Chen. Towards a better understanding of software evolution: an empirical study on open-source software, J. Softw. Evol. Process 25 (3) (Mar.2013) 193–218.
- [60] T. Kaur, N. Ratti, P. Kaur, Applicability of Lehman laws on open source evolution: a case study, Int. J. Comput. Appl. 93 (18) (May2014) 40–46.
- [61] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski. Metrics and laws of software evolution - The nineties view. In Proc. of the Fourth Intl. Software Metrics Symposium, Albuquerque, NM, November 1997.
- [62] Fred Brooks. The Mythical Man-Month. Addison-Wesley. 1975 & 1995. ISBN 0-201-00650-2 & ISBN 0-201-83595-9.
- [63] ISO/IEC 14764:2006, 2006.

[64] W.M. Turski, The reference model for smooth growth of software systems revisited, IEEE Trans. Softw. Eng. 28 (8) (Aug.2002) 814–815.

[65] D.J. Sheskin, D. Sheskin, Handbook of Parametric and Nonparametric Statistical Procedures, Second Edition, 2 ed., Chapman and Hall/CRC, Boca Raton, 2000.

[66] <https://docs.sonarqube.org/display/SONAR/Metric+Definitions>. SonarQube Documentation. Sep 05, 2017.

[67] Stephen Haunts. <https://stephenhaunts.com/2013/02/18/unit-test-coverage-code-metrics-and-static-code-analysis/>. UNIT TEST COVERAGE, CODE METRICS, AND STATIC CODE ANALYSIS. FEBRUARY 18, 2013