

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Δυνατότητες Επαναχρησιμοποίησης Λογισμικού σε Έργα JavaScript

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΤΕΡΖΗ ΑΝΑΣΤΑΣΙΑΣ

Επιβλέπων: Μπίμπη Σταματία

Επίκουρη Καθηγήτρια

ΚΟΖΑΝΗ/ΙΟΥΛΙΟΣ/2021

ΑΥΤΗ Η ΣΕΛΙΔΑ ΕΙΝΑΙ ΣΚΟΠΙΜΑ ΛΕΥΚΗ



HELLENIC DEMOCRACY
UNIVERSITY OF WESTERN MACEDONIA
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL
& COMPUTER ENGINEERING

ece.uowm.gr

Reuse Opportunities in JavaScript applications

THESIS

TERZI ANASTASIA

SUPERVISOR: Bibi Stamatia

Assistant Professor

KOZANI/JULY/2021

ΑΥΤΗ Η ΣΕΛΙΔΑ ΕΙΝΑΙ ΣΚΟΠΙΜΑ ΛΕΥΚΗ



ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο “**Δυνατότητες Επαναχρησιμοποίησης Λογισμικού σε Έργα JavaScript**” καθώς και τα ηλεκτρονικά αρχεία και πηγαίνοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κα. **Μπίμπη Σταματία** αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Copyright (C) Ονοματεπώνυμο Φοιτητή & Επιβλέποντα, Έτος, Πόλη

Copyright (C) Τερζή Αναστασία, Μπίμπη Σταματία, 2021, Κοζάνη

Υπογραφή Φοιτητή: _____

ΑΥΤΗ Η ΣΕΛΙΔΑ ΕΙΝΑΙ ΣΚΟΠΙΜΑ ΛΕΥΚΗ

Περίληψη

Η JavaScript αποτελεί σήμερα μια από τις πιο χρησιμοποιούμενες γλώσσες προγραμματισμού διαδικτυακών εφαρμογών. Η πλειοψηφία των εφαρμογών JavaScript κάνει χρήση λογισμικού τρίτων είτε με μορφή βιβλιοθηκών είτε με μορφή plug-in ώστε να εκμεταλλευτεί και να ενσωματώσει λειτουργίες που δεν παρέχει άμεσα. Μέρος της πτυχιακής εργασίας αποτελεί η μελέτη δημοφιλών επαναχρησιμοποιούμενων στοιχείων και η διερεύνηση του τύπου πρόσθετης λειτουργικότητας που επιλέγεται περισσότερο. Επιπλέον εξετάζεται η προτίμηση των προγραμματιστών απέναντι στις εκδόσεις του τρίτου λογισμικού καθώς και πως η σχέση μεταξύ των συνυπαρχόντων ζευγών λογισμικού επηρεάζει την επιλογή αυτή. Στο πλαίσιο της πτυχιακής έγινε μελέτη μεταξύ 129 στοιχείων τα οποία επαναχρησιμοποιούνται 9389 φορές, από τα 430 πιο δημοφιλή JavaScript έργα του GitHub. Τα αποτελέσματα δείχνουν ότι οι Compilers και τα Testing Units είναι οι πιο συνηθισμένοι τύποι λειτουργιών που επαναχρησιμοποιούνται είτε μεμονωμένα είτε σε ζεύγη, ενώ η πλειονότητα των εφαρμογών τείνουν να υιοθετούν τις πρόσφατες εκδόσεις των επαναχρησιμοποιούμενων στοιχείων.

Λέξεις Κλειδιά

JavaScript, Επαναχρησιμοποίηση, Λειτουργικότητα, Τρίτο Λογισμικό , Σχέση Συνύπαρξης

Abstract

JavaScript nowadays is among the most popular programming languages, used for developing web and IoT applications. Currently, the majority of JavaScript applications is reusing third-party components to acquire various functionalities. In this paper we isolate popular reused components and explore the type of functionality that is mostly being reused. Additionally, we examine whether the client applications adapt to the most recent versions of the reused components, and further study the reuse intensity of pairs of components that coexist in client applications. For this purpose, we performed a case study on 129 components reused 9389 times by 430 JavaScript applications hosted in GitHub. The results show that Compiler and Testing Units are the most common types of functionality being reused both individually and as pair, while the majority of client applications tend to adopt the recent versions of the reused components.

Keywords

Component, JavaScript, Reuse, Reuse Intensity

Ευχαριστίες

Ευχαριστώ την οικογένεια μου που πριν 5 χρόνια μου έδωσε την ευκαιρία να κάνω αυτό το βήμα και 5 χρόνια τώρα στηρίζει την προσπάθεια μου.

Στον αδερφό μου που αποτέλεσε τον πρώτο και πιο πολύτιμο αναγνώστη και επιτηρητή της εργασίας.

Ιδιαίτερες ευχαριστίες στην επιβλέπουσα κα Μπίμπη για όλες τις ευκαιρίες που μου έδωσε κατά τη διάρκεια της φοίτησης μου και για την εξαιρετική συνεργασία μας σε ένα ακόμα έργο.

Τέλος ευχαριστώ όσους βοήθησαν λίγο ή πολύ με τις ιδέες τους, τις παρατηρήσεις τους και τις γνώσεις τους για την παραγωγή αυτού του έργου

ΚΟΖΑΝΗ/ΙΟΥΛΙΟΣ/2021

ΑΥΤΗ Η ΣΕΛΙΔΑ ΕΙΝΑΙ ΣΚΟΠΙΜΑ ΛΕΥΚΗ

Περιεχόμενα

Περίληψη	7
Abstract	8
Ευχαριστίες	9
Περιεχόμενα	11
Κατάλογος Εικόνων	13
Κατάλογος Πινάκων	14
Πρόλογος	15
Κεφάλαιο 1: Εισαγωγή	17
1.1 Αντικείμενο της διπλωματικής	18
1.2 Οργάνωση του τόμου	18
Κεφάλαιο 2: Θεωρητικό υπόβαθρο	20
2.1 Υπάρχουσα Βιβλιογραφία	20
2.1.1 Πώς ορίζουμε την επαναχρησιμοποίηση λογισμικού	20
2.1.2 Γιατί Έργα JavaScript	22
2.1.3 Εφαρμογές JavaScript	22
2.1.4 Αντιμετώπιση Χρήσης Τρίτου Λογισμικού	23
2.1.5 Ασφάλεια Χρήσης τρίτου Λογισμικού	23
2.1.6 Αναβάθμιση και Εξέλιξη Έργου	25
Κεφάλαιο 3: Ανάλυση και Σχεδίαση Θέματος	27
3.1 Στόχος και Ερευνητικά Ερωτήματα	27
3.2 Ανάλυση Μελέτης Περίπτωσης	28
3.2.1 Συγκέντρωση και Ανάλυση δεδομένων	28
3.2.2 Μέθοδοι Ανάλυσης	33
Κεφάλαιο 4: Υλοποίηση	36
4.1 Λεπτομέρειες υλοποίησης	36
Κεφάλαιο 5: Αποτελέσματα	41
5.1.1 Ερευνητικό ερώτημα 1	41
5.1.2 Ερευνητικό ερώτημα 2	49
5.1.3 Ερευνητικό ερώτημα 3	53
Κεφάλαιο 6: Συμπεράσματα και Μελλοντική Επέκταση Έργου	58

6.1 Συμπεράσματα	58
6.2 Μελλοντικές Επεκτάσεις	59
Κεφάλαιο 7: Επίλογος	60
7.1 Απειλές	60
7.2 Σημασία Έργου	60
Παραρτήματα	62
A. Bash Script	62
B. Μορφή Csv	66
C. Αρχεία για επανάληψη μελέτης	67
Βιβλιογραφία	69
Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια	71
Απόδοση ξενόγλωσσων όρων	72

Κατάλογος Εικόνων

ΕΙΚΟΝΑ 1: ΠΑΡΑΔΕΙΓΜΑ ΓΡΑΦΟΥ ΕΠΑΝΑΧΡΗΣΙΜΟΠΟΙΗΣΗΣ ΠΑΚΕΤΩΝ ΑΠΟ ΕΡΓΑ ΛΟΓΙΣΜΙΚΟΥ .	31
ΕΙΚΟΝΑ 2: ΠΑΡΑΔΕΙΓΜΑ ΑΡΧΕΙΟΥ PACKAGE.JSON.	37
ΕΙΚΟΝΑ 3: ΠΑΡΑΔΕΙΓΜΑ ΑΡΧΕΙΟΥ PACKAGE.JSON ΜΕΤΑ ΤΗΝ ΕΠΕΞΕΡΓΑΣΙΑ.	38
ΕΙΚΟΝΑ 4: ΔΙΑΓΡΑΜΜΑΤΙΚΗ ΑΠΕΙΚΟΝΙΣΗ ΔΙΑΔΙΚΑΣΙΑΣ SCRIPT.	39
ΕΙΚΟΝΑ 5: ΚΑΜΠΥΛΗ ΛΗΨΗΣ TESTING UNITS ΔΙΑΣΤΗΜΑ 5 ΕΤΩΝ.	43
ΕΙΚΟΝΑ 6: ΚΑΜΠΥΛΗ ΛΗΨΗΣ COMPILERS ΔΙΑΣΤΗΜΑ 5 ΕΤΩΝ.	43
ΕΙΚΟΝΑ 7: ΚΑΜΠΥΛΗ ΛΗΨΗΣ USER INTERFACES ΔΙΑΣΤΗΜΑ 5 ΕΤΩΝ.	44
ΕΙΚΟΝΑ 8: ΚΑΜΠΥΛΗ ΛΗΨΗΣ FRAMEWORKS ΔΙΑΣΤΗΜΑ 5 ΕΤΩΝ.	44
ΕΙΚΟΝΑ 9: ΚΑΜΠΥΛΗ ΛΗΨΗΣ INTEROPERABILITY UNITS ΔΙΑΣΤΗΜΑ 5 ΕΤΩΝ.	45
ΕΙΚΟΝΑ 10: ΓΡΑΦΗΜΑ ΚΑΤΑΝΟΜΗΣ ΕΚΔΟΣΕΩΝ ΤΡΙΤΟΥ ΛΟΓΙΣΜΙΚΟΥ.	49
ΕΙΚΟΝΑ 11: ΧΑΡΤΗΣ ΘΕΡΜΟΤΗΤΑΣ.	55
ΕΙΚΟΝΑ 12: ΣΤΙΓΜΙΟΤΥΠΟ BASH SCRIPT BULLETS 1-4	63
ΕΙΚΟΝΑ 13: ΣΤΙΓΜΙΟΤΥΠΟ BASH SCRIPT BULLET 5	63
ΕΙΚΟΝΑ 14: ΣΤΙΓΜΙΟΤΥΠΟ BASH SCRIPT BULLETS 6-7	64
ΕΙΚΟΝΑ 15: ΣΤΙΓΜΙΟΤΥΠΟ BASH SCRIPT BULLETS 8-9	64
ΕΙΚΟΝΑ 16: ΣΤΙΓΜΙΟΤΥΠΟ BASH SCRIPT BULLETS 1-2	65
ΕΙΚΟΝΑ 17: ΣΤΙΓΜΙΟΤΥΠΟ ΜΕΜΟΝΩΜΕΝΟΥ CSV	66
ΕΙΚΟΝΑ 18: ΣΤΙΓΜΙΟΤΥΠΟ ΤΕΛΙΚΟΥ CSV	66
ΕΙΚΟΝΑ 19: ΠΕΡΙΕΧΟΜΕΝΟ REPOSITORY	67
ΕΙΚΟΝΑ 20: ΠΕΡΙΕΧΟΜΕΝΟ ΦΑΚΕΛΟΥ LV_CSV	68
ΕΙΚΟΝΑ 21: ΠΕΡΙΕΧΟΜΕΝΟ ΥΠΟΦΑΚΕΛΩΝ LV_CSV	68

Κατάλογος Πινάκων

Πίνακας 1: Απεικόνισης Χρήσης Εργαλείων-Μεθόδων ανά Ερευνητικό Ερώτημα.	34
Πίνακας 2: Απεικόνισης Χρήσης Μεταβλητών ανά Μέθοδο και Ερευνητικό Ερώτημα.	35
Πίνακας 3: Απεικόνισης Χρήσης Μεταβλητών ανά Σχέση .	35
Πίνακας 4: Απεικόνιση Δημοφιλέστερων Στοιχείων JavaScript στα 430 έργα	41
Πίνακας 5: Απεικόνιση Πλήθους Χρήσης Τρίτου Λογισμικού ανά τύπο Λειτουργικότητας	45
Πίνακας 6: Απεικόνιση Πλήθους Εξαρτήσεων από και προς Τρίτο Λογισμικό για τα 35 πιο Χρησιμοποιούμενα Στοιχεία	46
Πίνακας 7: Απεικόνιση Πλήθους Εξαρτήσεων ανά Είδος για τα 35 πιο Χρησιμοποιούμενα Στοιχεία	47
Πίνακας 8: Περισσότερο Χρησιμοποιούμενη Έκδοση από τα 35 πιο Δημοφιλή Στοιχεία	50
Πίνακας 9: Απεικόνιση και Περιγραφή Σχέσης Ζευγών από Ανάλυση Spearman	53
Πίνακας 10: Απεικόνιση Δημοτικότητας ανά ζεύγος στοιχείων τρίτου μέρους	55

Πρόλογος

Η επαναχρησιμοποίηση λογισμικού αποτελεί ένα από τα μεγαλύτερα κινήματα της σύγχρονης εποχής. Επικρατεί μια τάση απόδοσης νέων λειτουργιών σε πακέτα και μέρη τρίτου λογισμικού που δεν εξυπηρετούν πλέον τον αρχικό σκοπό χρήσης τους, είτε λόγω νέων τεχνολογιών είτε λόγω δυσκολίας χρήσης. Σκοπός του κινήματος επαναχρησιμοποίησης ή υρεγκλίας είναι η επαναχρησιμοποίηση των υλικών με τέτοιο τρόπο ώστε να δημιουργηθεί ένα προϊόν υψηλότερης ποιότητας ή αξίας από το πρωτότυπο. Επιπλέον με την χρήση ήδη υπαρχόντων μερών λογισμικού, το κίνημα εστιάζει στην μείωση του κόστους ανάπτυξης και την αύξηση της παραγωγικότητας των προγραμματιστών. Το κίνημα είναι ευρέως διαδεδομένο, συναντάται ως επαναχρησιμοποίηση κώδικα ή αυτόνομων μονάδων λογισμικού κατά την δημιουργία νέων έργων και αποτελεί μια από τις πιο συνηθισμένες μεθόδους για την παραγωγή λογισμικού. Πρόκειται για μια τάση ικανή να ακολουθήσει την ταχύτατη αλλαγή σε απαιτήσεις χρηστών και την διαρκή εξέλιξη των προσφερόμενων τεχνολογιών. Η ανάγκη για παραγωγή περισσότερων έργων σε μικρό χρονικό διάστημα και με μειωμένο κόστος παραγωγής την καθιστά μια από τις πιο δημοφιλείς μεθόδους μαζικής παραγωγής λογισμικού.

Στην εποχή που οι εφαρμογές ιστού αποτελούν αναπόσπαστο κομμάτι της ζωής μας και η ανάγκη για γρήγορη ανάπτυξη και προσαρμογή στις νέες τεχνολογίες είναι πιο επιτακτική από ποτέ, η επαναχρησιμοποίηση λογισμικού αποτελεί την πιο ρεαλιστική λύση [16]. Η JavaScript αποτελεί μία από τις πιο διαδεδομένες γλώσσες προγραμματισμού για την ανάπτυξη εφαρμογών ιστού [5], τόσο στην πλευρά του πελάτη όσο και στη πλευρά του εξυπηρετητή. Η ευρεία και διαδεδομένη χρήσης της γλώσσας για την ανάπτυξη εφαρμογών client και server, που συνεπάγεται με την αναγκαιότητα χρήσης δομών διασύνδεσης και απλοποίησης διαδικασίας συγγραφής κώδικα αποτελώντας μια ιδανική βάση για να ξεκινήσουμε την μελέτη.

Στο πλαίσιο αυτής της εργασίας, πραγματοποιήθηκε μελέτη περίπτωσης σε 129 στοιχεία που επαναχρησιμοποιήθηκαν 9389 φορές, από τις 430 πιο δημοφιλείς ως προς τα forks, εφαρμογές JavaScript που φιλοξενούνται στο αποθετήριο του GitHub. Οι τρεις άξονες γύρω από τους οποίους κινήθηκε η έρευνα είναι:

- α) ο εντοπισμός των δημοφιλέστερων επαναχρησιμοποιούμενων στοιχείων και η διερεύνηση της λειτουργικότητας που προσφέρουν,
- β) η προτίμηση προς την έκδοση των στοιχείων τρίτων μερών και
- γ) ο τρόπος συνδυασμού των στοιχείων και η ένταση συσχέτισης των ζευγών τρίτου μέρους.

Από την μελέτη έγινε αντιληπτό ότι η συχνότητα επαναχρησιμοποίησης των στοιχείων εξαρτάται από την λειτουργικότητα που προσφέρουν στην παραγόμενη εφαρμογή JavaScript. Τα αποτελέσματα έδειξαν ότι τα στοιχεία που ανήκουν στις κατηγορίες Compilers, τα Testing Units και τα Interoperability Units κυριαρχούν στην επιλογή των προγραμματιστών είτε μεμονωμένα είτε σε ζεύγη. Στα έργα που χρησιμοποιούν μεγάλο πλήθος στοιχείων τρίτου λογισμικού, οι δημιουργοί τείνουν να χρησιμοποιούν παλαιότερες εκδόσεις ενώ στην αντίθετη περίπτωση όταν το πλήθος των στοιχείων

τρίτου λογισμικού είναι μικρό, οι προγραμματιστές μεταβαίνουν συνεχώς στην τελευταία έκδοση. Η ίδια αντιμετώπιση υπάρχει και προς το πλήθος εξαρτήσεων. Τέλος για την σχέση μεταξύ των ζευγών, καταλήξαμε στο συμπέρασμα ότι τα δημοφιλή ζευγάρια δεν δημιουργούνται μόνο χάρη στον καλό συνδυασμό των λειτουργιών που προσφέρουν αλλά επηρεάζονται και από τις εξαρτήσεις των στοιχείων προς άλλα πακέτα.

Κεφάλαιο 1: Εισαγωγή

Αναμφίβολα, η JavaScript αποτελεί μία από τις πιο διαδεδομένες γλώσσες προγραμματισμού για την ανάπτυξη εφαρμογών ιστού [5]. Μια πρόσφατη έρευνα σε περισσότερους από 17.000 προγραμματιστές, σε 159 χώρες, μεταξύ Νοεμβρίου 2019 και Φεβρουαρίου 2020, έδειξε ότι πάνω από 12,2 εκατομμύρια προγραμματιστές χρησιμοποιούν αυτήν τη στιγμή JavaScript σε όλο τον κόσμο, καθιστώντας τη γλώσσα ως την πιο δημοφιλή [4]. Η ανάπτυξη και διάδοση της JavaScript οφείλεται στο γεγονός ότι είναι μια ελαφριά, δυναμική γλώσσα, που μπορεί να χρησιμοποιηθεί τόσο για front-end σχεδιασμό όσο και για back-end ανάπτυξη, καθιστώντας την έτσι μια παγκόσμια αποδεκτή γλώσσα προγραμματισμού. Επιπλέον υπάρχουν ολοένα και περισσότερα στοιχεία που αποδεικνύουν ότι η ανάπτυξη κώδικα στη γλώσσα προγραμματισμού JavaScript γίνεται πιο κατανεμημένη και συνεργατική [16]. Οι προγραμματιστές έχουν πρόσβαση σε πληθώρα διαθέσιμων πακέτων λογισμικού ανοιχτού κώδικα που μπορούν να χρησιμοποιηθούν ελεύθερα για να προσθέτουν έτοιμες λειτουργίες στις εφαρμογές τους και να απολαμβάνουν τα οφέλη της επαναχρησιμοποίησης λογισμικού.

Η επαναχρησιμοποίηση λογισμικού σύμφωνα με τον McIlroy [15], είναι «η διαδικασία δημιουργίας συστημάτων λογισμικού από υπάρχον λογισμικό και όχι η κατασκευή συστημάτων λογισμικού από το μηδέν». Τα οφέλη που προσφέρει η επαναχρησιμοποίηση λογισμικού περιλαμβάνουν:

- το ελαχιστοποιημένο κόστος ανάπτυξης,

Με την χρήση έτοιμων πακέτων και προσαρτώμενων λειτουργιών ο προγραμματιστής εξοικονομεί χρόνο από την ανάπτυξη λογισμικού και η διαδικασία παραγωγής λογισμικού επιταχύνεται.

- την αυξημένη αποδοτικότητα,

Πλέον ο προγραμματιστής διοχετεύει τον χρόνο και την δημιουργικότητα του, για να επιλύσει πιο εξειδικευμένα και συγκεκριμένα προβλήματα που υπάρχουν στο έργο χωρίς να αναλώνεται σε διαδικασίες ασήμαντες και εύκολα επιλύσιμες.

- την δυναμική συντήρηση και τη

Η χρήση τρίτου λογισμικού περιορίζει την ανάγκη να επέμβει ο προγραμματιστής σε μεγάλο μέρους του κώδικα για να κάνει αλλαγές. Αρκεί, ο ίδιος, να κάνει αναβαθμίσεις των τρίτων μερών και να επεμβαίνει μόνο στο κομμάτι ενσωμάτωσης του τρίτου κώδικα στον δικό του.

- βελτιωμένη ποιότητα [12].

Η ποιότητα αφορά την ανάπτυξη του λογισμικού που πλέον γίνεται με μικρότερη πιθανότητα εμφάνισης προβλημάτων κακής συγγραφής, καθώς ο κώδικας είναι αρθρωτός και αποτελείται από διαφορετικά αυτόνομα κομμάτια και όχι από μεγάλα κομμάτια κώδικα που παρήχθησαν από το ίδιο άτομο. Επιπλέον το γεγονός ότι είναι καλώς ελεγμένα, αυτόνομα και παρέχουν βιβλιογραφία τα καθιστά πιο αξιόπιστα όπως και κατ' επέκταση και τον ίδιο το κώδικα εφαρμογής.

Όσον αφορά την ανάπτυξη εφαρμογών JavaScript, υπάρχει ήδη διαθέσιμη μια σειρά πακέτων διαχείρισης εξαρτήσεων έτοιμων βιβλιοθηκών (πρω, nexus, Github) που μπορούν να διευκολύνουν την επαναχρησιμοποίηση των στοιχείων .js. Παρά το γεγονός αυτό, η πρόκληση επιλογής της σωστής λειτουργικότητας προς επαναχρησιμοποίηση, όπως και ο καθορισμός των κατάλληλων στοιχείων που θα συνθέσουν την νέα εφαρμογή παραμένει.

1.1 Αντικείμενο της διπλωματικής

Στο πλαίσιο της παρούσας μελέτης, αντιμετωπίζουμε αυτήν την πρόκληση διερευνώντας τα διαθέσιμα στοιχεία JavaScript που μπορούν να επαναχρησιμοποιηθούν εξετάζοντας παράλληλα τις λειτουργίες που προσφέρουν. Στόχος μας είναι να αναγνωρίσουμε τις δυνατότητες επαναχρησιμοποίησης που προσφέρονται από τις βιβλιοθήκες και τα πακέτα ανοιχτού κώδικα τα οποία είναι υλοποιημένα σε JavaScript. Για το σκοπό αυτό, πραγματοποιήσαμε μια μελέτη περίπτωσης σε 129 στοιχεία JavaScript που ανακτήθηκαν από το αποθετήριο του GitHub. Στόχος της μελέτης περίπτωσης είναι να διερευνηθεί η:

- **Λειτουργικότητα στοιχείων JavaScript:** Είναι γνωστό ότι η επαναχρησιμοποίηση λογισμικού εκτελείται αποτελεσματικότερα και συχνότερα μεταξύ εφαρμογών ίδιου αντικειμένου [17], συνεπώς θα μελετήσουμε τη σχέση μεταξύ της συχνότητας επαναχρησιμοποίησης και της λειτουργίας που προσφέρουν τα στοιχεία τρίτου λογισμικού. Η περιγραφή των λειτουργιών των στοιχείων που μελετώνται προέρχεται από τις επίσημες ιστοσελίδες ανάπτυξης τους. Οι λειτουργίες που μελετήθηκαν περιλαμβάνουν: Μεταγλωττιστές(Compilers), Πλαίσια ανάπτυξης(Frameworks), Μονάδες ελέγχου(Testing Units), Μονάδες διεπαφής χρήστη(User Interface Units) και Μονάδες διασύνδεσιμότητας (Interoperability Units).
- **Ένταση συνύπαρξης ζευγών στοιχείων JavaScript (intensity of coexistence pairs):** Με σκοπό την πλήρη αξιοποίηση των πλεονεκτημάτων της επαναχρησιμοποίησης οι προγραμματιστές τείνουν να χρησιμοποιούν ταυτόχρονα μια ποικιλία στοιχείων που εξυπηρετούν διαφορετικούς σκοπούς. Με τον προσδιορισμό των ζευγών τρίτου λογισμικού που συχνά χρησιμοποιούνται στο πλαίσιο ανάπτυξης μίας εφαρμογής θα παραχθεί ένα συμπέρασμα ως προς την αλληλοσυμπλήρωση των λειτουργιών των μονάδων λογισμικού

1.2 Οργάνωση του τόμου

Το υπόλοιπο έγγραφο οργανώνεται ως εξής:

- Στην Ενότητα 2, μέσα από παραδείγματα βιβλιογραφίας παρουσιάζεται το θεωρητικό υπόβαθρο της έρευνας
- Στην Ενότητα 3, αναλύεται η μελέτη περίπτωσης και γίνεται αναφορά στα ερευνητικά ερωτήματα που απασχολούν την εργασία.
- Στην Ενότητα 4, περιγράφεται ο τρόπος υλοποίησης των εργαλείων που χρησιμοποιήθηκαν στη μελέτη και η πορεία που ακολουθήθηκε για την εξαγωγή αποτελεσμάτων.

- Στην Ενότητα 5, παρουσιάζονται και ερμηνεύονται τα αποτελέσματα ανά ερευνητικό ερώτημα.
- Στην Ενότητα 6, γίνεται συζήτηση των συμπερασμάτων μαζί με τις μελλοντικές επεκτάσεις της παρούσας μελέτης.
- Στην Ενότητα 7, αναφέρονται οι απειλές για την εγκυρότητα της μελέτης, καθώς και ο τρόπος με τον οποίο μπορούν να αξιοποιηθούν μελλοντικά τα συμπεράσματα της μελέτης από ερευνητές και τεχνολόγους λογισμικού.
- Τέλος, στα Παραρτήματα, ο αναγνώστης θα βρει τον κώδικα και τα αρχεία που χρησιμοποιήσαμε για να επαναλάβει την μελέτη μας

Κεφάλαιο 2: Θεωρητικό υπόβαθρο

Σε αυτό το τμήμα πραγματοποιείται βιβλιογραφική ανασκόπηση του υπό διαπραγμάτευση θέματος. Καθορίζονται οι βασικοί όροι και έννοιες που θα αναλυθούν στο πλαίσιο της μελέτης και διατυπώνονται οι ερευνητικές υποθέσεις που πρόκειται να διερευνηθούν. Στόχος του τμήματος είναι η ενημέρωση του αναγνώστη προς την έρευνα που έχει ήδη διενεργηθεί αναφορικά με το θέμα της εργασίας, δίνει τη δυνατότητα στον αναγνώστη που δεν γνωρίζει πολλά πράγματα σχετικά με το υπό εξέταση θέμα, αφενός να ενημερωθεί για τις υπάρχουσες προσεγγίσεις σε αυτό, αφετέρου, να μπορέσει να παρακολουθήσει και να κρίνει την ερευνητική προσέγγιση που ακολουθείται στη συνέχεια.

2.1 Υπάρχουσα Βιβλιογραφία

2.1.1 Πως ορίζουμε την επαναχρησιμοποίηση λογισμικού

Όπως αναφέραμε ήδη στον πρόλογο, άξονας της παρούσας εργασίας αποτελεί η επαναχρησιμοποίηση έτοιμων συνιστωσών κώδικα σε έργα λογισμικού. Η επαναχρησιμοποίηση αποτελεί μια από τις βασικότερες στρατηγικές επίλυσης προβλημάτων στις περισσότερες ανθρώπινες δραστηριότητες και η ανάπτυξη λογισμικού δεν αποτελεί εξαιρεση [16]. Η επαναχρησιμοποίηση λογισμικού φαίνεται ως η πλέον ρεαλιστική προσέγγιση για την αύξηση του κέρδους, τη βελτίωση της παραγωγικότητας και της ποιότητας προϊόντων λογισμικού, που αναζητά η βιομηχανία λογισμικού [16]. Επαναχρησιμοποίηση λογισμικού σημαίνει χρήση των εισόδων, των διαδικασιών, των μεθόδων και των αποτελεσμάτων που προέκυψαν από προηγούμενες προσπάθειες ανάπτυξης λογισμικού. Βασίζεται στην προϋπόθεση ότι η αναζήτηση λύσης από μηδενική βάση συνεπάγεται περισσότερη προσπάθεια από την ενσωμάτωση και επέκταση λύσης που έχει αποδοθεί σε παρόμοιο πρόβλημα. Στόχος της αποτελεί η κατασκευή νέου λογισμικού που μπορεί με τη σειρά του να επαναχρησιμοποιηθεί από το σχεδιασμό και την κατασκευή άλλου λογισμικού. Συνεπώς πρόκειται για μια αέναη κυκλική διαδικασία. Η επαναχρησιμοποίηση λογισμικού χωρίζεται σε δύο κατηγορίες:

α) **λευκού κουτιού, white box** και Η πρώτη αναφέρεται στην επαναχρησιμοποίηση λογισμικού με γνώση του προγραμματιστή. Ο προγραμματιστής σε αυτή την περίπτωση γνωρίζει και έχει επιλέξει τα πακέτα και το τρίτο λογισμικό που χρησιμοποιεί στην εφαρμογή του. Πρόκειται για την άμεση χρήση τρίτου λογισμικού.

β) **μαύρου κουτιού, black box.** Η μέθοδος black box αφορά την χρήση τρίτου λογισμικού μέσω των εξαρτήσεων προς αυτό και χωρίς να γνωρίζουμε πως είναι διαμορφωμένος ο κώδικάς του. Μπορεί ο προγραμματιστής να έχει επιλέξει να χρησιμοποιήσει ένα πακέτο για τις ανάγκες του έργου αλλά οι εξαρτήσεις του ίδιου του πακέτου καθώς και τα τρίτα πακέτα που εκείνο χρησιμοποιεί δημιουργούν μια σχέση black box.

Οι δύο αυτές τεχνικές χρήσης τρίτου λογισμικού είναι εξίσου διαδεδομένες και δεν υπάρχει κάποια συγκεκριμένη μεθοδολογία για τη χρήση τρίτου λογισμικού που να έχει επικρατήσει. Η επαναχρησιμοποίηση τρίτου λογισμικού αφορά κυρίως σε πακέτα και βιβλιοθήκες ανοιχτού λογισμικού, που προσφέρουν τις λειτουργίες που αναζητά ο χρήστης με χαμηλό ή μηδενικό κόστος. Το κόστος αναφέρεται τόσο στο οικονομικό, όσο

και στο κόστος διασύνδεσης και συντήρησης του προσαρτώμενου τρίτου μέρους. Για να κατανοήσει ο αναγνώστης την σημασία μικρού κόστους συντήρησης αρκεί να αναφέρουμε ότι κατά μέσω όρο σε έργα Java επαναχρησιμοποιούνται 70 πακέτα ανοιχτού λογισμικού [17].

Τα στοιχεία που καθορίζουν την ποιότητα του τρίτου λογισμικού ως προς την επαναχρησιμοποίηση του είναι [9]:

- **Επαναχρησιμοποίηση (Reuse):** υποδεικνύει τον βαθμό στον οποίο ένα προϊόν λογισμικού βασίζεται σε επαναχρησιμοποιούμενα στοιχεία.
- **Προσαρμοστικότητα (Adaptability):** αντικατοπτρίζει την ευκολία με την οποία ένα στοιχείο μπορεί να προσαρμοστεί όταν επαναχρησιμοποιηθεί σε ένα νέο σύστημα. Η προσαρμοστικότητα επηρεάζεται από τρεις πτυχές:
 1. τη γλώσσα προγραμματισμού
 2. τη δομή του επαναχρησιμοποιούμενου στοιχείου και
 3. την ύπαρξη μεθόδων και διεπαφών για την επαναχρησιμοποίηση του στοιχείου.
- **Κόστος (Price):** δείχνει πόσο ακριβό δηλαδή δύσκολο, είναι να επαναχρησιμοποιηθεί .
- **Συντηρησιμότητα (Maintainability):** αντιπροσωπεύει το βαθμό στον οποίο ένα στοιχείο μπορεί να επεκταθεί, αφού προστεθεί στο σύστημα (π.χ. κατά τη διάρκεια μιας νέας έκδοσης).
- **Ποιότητα (Quality):** περιγράφει την εκπλήρωση των απαιτήσεων του στοιχείου:
 1. τα σφάλματα που εισάγει
 2. τις δοκιμαστικές περιπτώσεις που το επαληθεύουν
 3. την αξιολόγηση του από τους χρήστες .
- **Διαθεσιμότητα (Availability):** περιγράφει πόσο εύκολο είναι να βρεθεί ένα στοιχείο (π.χ., αμέσως μετά την αναζήτηση, μη διαθέσιμο κ.λπ.).
- **Βιβλιογραφία (Documentation):** Η υπάρχουσα βιβλιογραφία που συνοδεύει το τρίτο λογισμικό ώστε η λειτουργία του και οι απαιτήσεις του να γίνουν πιο εύκολα κατανοητές στον προγραμματιστή και άρα η χρήση του πιο εύκολη.
- **Πολυπλοκότητα (Complexity):** Αφορά την εσωτερική δομή του εξαρτήματος και απεικονίζεται σε πολλές πτυχές της ποιότητας (π.χ. την ευκολία κατανόησης και προσαρμογής σε ένα νέο πλαίσιο).

2.1.2 Γιατί Έργα JavaScript

Η JavaScript έχει εξελιχθεί σε μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού και χρησιμοποιείται εκτενώς για τη δημιουργία εφαρμογών ιστού, τόσο στην πλευρά του πελάτη όσο και στη πλευρά του εξυπηρετητή. Ένα ακόμα ιδιαίτερο χαρακτηριστικό της είναι ότι δεν υποστηρίζει μόνο ένα προγραμματιστικό στιλ, αλλά επιτρέπει στον δημιουργό να χρησιμοποιήσει κατά την ανάπτυξη εφαρμογών, αντικειμενοστρέφεια και στοιχεία δομημένου προγραμματισμού. Η δυική αυτή φύση της και οι δυνατότητες που προσφέρει για χρήση τόσο στο κομμάτι της front-end όσο και της back-end ανάπτυξης, αποτέλεσε κίνητρο εκτεταμένης μελέτης.

Μεγάλο μέρος της βιβλιογραφίας επικεντρώνεται στον τρόπο ανάπτυξης εφαρμογών, στα χαρακτηριστικά [8] και τη δυναμική της γλώσσας [4], ενώ μόνο ένα μικρό ποσοστό ερευνητών έχουν πραγματοποιήσει μελέτη πάνω στα στοιχεία από τα οποία απαρτίζονται οι εφαρμογές [1] και την ασφάλεια προς την χρήση τους [18]. Χρησιμοποιώντας ως αφορμή την τρέχουσα βιβλιογραφία προσπαθήσαμε να δημιουργήσουμε ένα προϊόν που θα συμπληρώνει την υπάρχουσα γνώση απαντώντας σε ερωτήματα που προϋπήρχαν ή γεννήθηκαν κατά τη διάρκεια ενασχόλησης μας.

2.1.3 Εφαρμογές JavaScript

Η αναζήτησή και η μελέτη μας ξεκίνησε από το ερώτημα “πως οι προγραμματιστές καταφέρνουν να συνδυάζουν τις δυνατότητες του δομημένου προγραμματισμού και της αντικειμενοστρέφειας”. Σε ένα δείγμα 70.000 έργων, οι ερευνητές παρατηρούν [8], ότι το μεγαλύτερο μέρος των εφαρμογών βασίζεται σε δομημένο προγραμματισμό και η χρήση αντικειμένων είναι περιορισμένη. Θεωρούν ότι το γεγονός αυτό οφείλεται στην παραδοσιακή χρήση της JavaScript, που περιοριζόταν στην χρήση του DOM που παρέχει ο browser και ότι η στάση αυτή θα αλλάξει όσο αυξάνονται οι απαιτήσεις των χρηστών. Συμπληρωματικά, οι συγγραφείς μας δίνουν την πληροφορία ότι το μικρό ποσοστό αντικειμένων που χρησιμοποιούνται είναι έτοιμα αντικείμενα που προσθέτουν λειτουργίες στην εφαρμογή και δεν συνθέτονται εξαρχής από τους προγραμματιστές. Επιπλέον, όπως αναφέρεται [8], οι προγραμματιστές JavaScript εφαρμογών χρησιμοποιούν συχνά δυσνόητους και ανορθόδοξους τρόπους γραφής κώδικα, γεγονός που οφείλεται αφενός στη δυσκολία της γλώσσας, αφετέρου στην έλλειψη εξειδίκευσης. Οι συγγραφείς εφαρμογών ιστού συχνά προέρχονται από διαφορετικό υπόβαθρο και δεν έχουν απαραίτητα προγούμενη επαφή με τον προγραμματισμό, ακόμα και αυτοί ωστόσο με εμπειρία φαίνεται να μη μπορούν να αντεπεξέλθουν απόλυτα στη δυαδική φύση της γλώσσας. Για να ξεπεράσουν αυτή την αδυναμία αλλά και για την διευκόλυνση τους καταφεύγουν στη χρήση τρίτων μερών λογισμικού.

2.1.4 Αντιμετώπιση Χρήσης Τρίτου Λογισμικού

Η χρήση του τρίτου λογισμικού ξεκινά αλλά δε περιορίζεται στην επίλυση δύσκολων και χρονοβόρων προγραμματιστικά προβλημάτων που συχνά απαιτούν ειδική γνώση. Οι συγγραφείς Abdalkareem et al.,[1] μέσα από τη μελέτη τους σε δείγμα 30.000 προ πακέτων και 38.000 έργων JavaScipt, μας ενημερώνουν πως οι προγραμματιστές τείνουν να χρησιμοποιούν τρίτο λογισμικό ακόμα και για πολύ μικρές και ασήμαντες διαδικασίες. Τα πακέτα με χαμηλή πολυπλοκότητα που χρησιμοποιούνται για την προσθήκη μικρών λειτουργιών χαρακτηρίζονται ως πακέτα μικρής κλίμακας (trivial packages), από τους συγγραφείς και αποτελούν το 11,3% εκ των 1.000 πιο δημοφιλών πακέτων του προ. Κομμάτι της μελέτης τους αποτέλεσε και ένα ερωτηματολόγιο προς 88 προγραμματιστές Node.js μέσω του οποίου οι συγγραφείς κατέγραψαν απόψεις προς την χρήση τρίτου λογισμικού. Ως αποτέλεσμα, μας εξηγούν ότι οι προγραμματιστές επιλέγουν την χρήση trivial πακέτων μιας και οι ίδιοι δεν αναλώνονται στην διαχείριση εύκολων διαδικασιών και με τον τρόπο αυτό αυξάνουν την αποδοτικότητα τους κατά την παραγωγή κώδικα εφαρμογής. Επιπλέον, οι ίδιοι θεωρούν τα πακέτα αυτά καλώς ελεγμένα και έτσι τα αντιμετωπίζουν ως πιο αξιόπιστη πηγή σε σχέση με τον κώδικα που παράγουν και ελέγχουν μόνοι τους. Οι συγγραφείς αναφέρουν ακόμα ότι οι προγραμματιστές με την χρήση πακέτων τρίτου λογισμικού επιδιώκουν μείωση πολυπλοκότητας συγγραφής κώδικα και ταυτόχρονη αύξηση στην αποδοτικότητα του [1]. Ωστόσο η επαναχρησιμοποίηση τρίτου λογισμικού δεν αποτελεί πανάκεια και σε πολλές περιπτώσεις αποδεικνύεται ότι εισάγει περισσότερα προβλήματα από όσα λύνει [1, 12, 20]. Συγκεκριμένα μπορεί η συγγραφή των πιο δύσκολων σημείων του κώδικα να γίνεται με μεγαλύτερη ακρίβεια και προσοχή, αλλά η ενσωμάτωση κώδικα τρίτου μέρους αποτελεί μια λεπτή διαδικασία που απαιτεί χρόνο και πιθανόν τελικά να καθυστερήσει την ανάπτυξη αντί να την απλοποιήσει. Το σημαντικότερο όμως πρόβλημα αφορά τις εξαρτήσεις που δημιουργούνται όταν οι προγραμματιστές στηρίζονται σε κώδικα τρίτων. Οι εξαρτήσεις αφορούν:

- α) εξάρτηση προς το άτομο που έχει δημιουργήσει και συντηρεί το τρίτο λογισμικό, καθώς πλέον ο προγραμματιστής πρέπει να λαμβάνει υπόψιν και τις αλλαγές που πραγματοποιεί ο δημιουργός τρίτου λογισμικού πριν προχωρήσει σε αναβάθμιση του δικού του κώδικα και
- β) τις εξαρτήσεις που το ίδιο το τρίτο μέρος έχει προς βιβλιοθήκες ή άλλα στοιχεία. Η κατάσταση αυτή είναι γνωστή ως dependency hell και συναντάται ευρέως στη βιβλιογραφία.

2.1.5 Ασφάλεια Χρήσης τρίτου Λογισμικού

Ιδιαίτερη εντύπωση από το παραπάνω άρθρο μας έκανε το γεγονός ότι οι προγραμματιστές εμπιστεύονται περισσότερο τον κώδικα τρίτων από τον κώδικα που γράφουν οι ίδιοι. Είναι όμως ασφαλής αυτή η προσέγγιση; Όσον αφορά την πηγή προέλευσης του κώδικα τρίτου μέρους, το προ αναφέρεται ως ένα περιβάλλον με πολλά προβλήματα ασφαλείας [20]. Το προ αποτελεί το πιο διαδεδομένο οικοσύστημα από το οποίο οι προγραμματιστές JavaScipt εφαρμογών προμηθεύονται τα πακέτα τρίτου μέρους που χρησιμοποιούν. Αρχικά, είναι σημαντικό να διευκρινίσουμε στον αναγνώστη ότι η JavaScipt δεν παρέχει κανένα είδος διαχωρισμού δικαιωμάτων μεταξύ κώδικα που φορτώνεται από διαφορετικά πακέτα και του κώδικα που παράγει ο ίδιος ο δημιουργός. Συνεπώς, οποιοδήποτε πακέτο τρίτων έχει πλήρη δικαιώματα σε ολόκληρη

την εφαρμογή. Για τον λόγο αυτό η πηγή προέλευσης του κώδικα τρίτου μέρους πρέπει να είναι απόλυτα αξιόπιστη και οι δημιουργοί και συντηρητές του τρίτου λογισμικού εξειδικευμένοι και περιορισμένοι. Βέβαια, όπως μας κάνουν γνωστό οι συγγραφείς, για την κατάθεση νέου πακέτου ή την αναβάθμιση σε υπάρχον, η μόνη προϋποθέσεις που πρέπει να τηρεί το άτομο είναι:

- a) να είναι εγγεγραμμένος χρήστης και
- β) να ανεβάσει σε έναν φάκελο που περιέχει συνοδευτικό αρχείο εξαρτήσεων, package.json.

Έπειτα αυτόματα το σύστημα καταγράφει το άτομο στην λίστα των συντηρητών αυτού του πακέτου. Περισσότερα από 600 δημοφιλή πακέτα του npm στηρίζονται σε τουλάχιστον 100 διαφορετικούς συντηρητές, με τα πιο δημοφιλή να στηρίζονται σε 10 και των μέσο όρο να βρίσκεται στους 40 συντηρητές. Η κατάσταση δε θα ήταν κρίσιμη αν υπήρχε μεγαλύτερος έλεγχος προς τα άτομα που μπορούν να χαρακτηριστούν συντηρητές. Για τους ερευνητές είναι πολύ ανησυχητικό το γεγονός ότι μόλις 391 συντηρητές έχουν πραγματοποιήσει αλλαγές σε περισσότερα από 100.000 πακέτα. Επιπλέον, οι συγγραφείς μας τονίζουν ότι κατά μέσο όρο κάθε πακέτο του npm έχει εξαρτήσεις προς αλλά 80 διαφορετικά πακέτα, με τα πιο δημοφιλή να έχουν περισσότερα από 100.000 εξαρτώμενα προς αυτά πακέτα. Για να γίνει αντιληπτή η έκταση του προβλήματος το 2016 το πακέτο left-pad αφαιρέθηκε από το npm προκαλώντας την κατάρρευση ενός μεγάλου ποσοστού πακέτων που είχαν άμεσες ή έμμεσες εξαρτήσεις προς αυτό. Ενώ το 2018 μέσα σε μία αναβάθμιση του eslint-scope, ενός πακέτου με μικρό αριθμό εξαρτήσεων προς αυτό, υπήρχε κακόβουλος κώδικας ο οποίος προκάλεσε βλάβη σε περίπου 1.000 πακέτα. Τα δύο αυτά περιστατικά τονίζουν την επικινδυνότητα εξάρτησης προς μεγάλο πλήθος τρίτου λογισμικού και πολλούς αγνώστους συντηρητές.

Μπορεί όντως η χρήση τρίτου λογισμικού να αποδειχτεί μοιραία αλλά είναι και απολύτως αναγκαία. Επισημαίνεται η σημασία χρήσης μονάδων ελέγχου, Testing Units, κατά τη δημιουργία του κώδικα και συνίσταται η αποφυγή χρήσης χειροκίνητων ελέγχων [7]. Στη μελέτη τους σε 373 έργα JavaScript, οι συγγραφείς, υπολόγισαν ότι μόνο το 60% των εφαρμογών έχουν πραγματοποιήσει έλεγχο για το κομμάτι κώδικα που απευθύνεται στον client, ενώ σε συντριπτικό ποσοστό, το 95% των εφαρμογών έχουν περάσει έλεγχο για το κομμάτι διασύνδεσης με τον server. Θεωρούν ότι η διαφορά στα ποσοστά οφείλεται στη δυσκολία δημιουργίας σεναρίων ελέγχου που να κάνουν trigger στα events που χρησιμοποιούν οι προγραμματιστές στη front-end ανάπτυξη. Για το κομμάτι διασύνδεσης χρήστη οι έλεγχοι που απαιτούνται αφορούν διαφορετικά κομμάτια της εφαρμογής (π.χ. κομμάτια οπτικοποίησης, διαχείρισης κινήσεων αφής ή ποντικιού κλπ) καθιστώντας τον ταυτόχρονο έλεγχο τους από δύσκολο έως αδύνατο. Για το λόγο αυτό τονίζουν ότι η χρήση τρίτου λογισμικού όχι μόνο πετυχαίνει μεγαλύτερη καλυψιμότητα κώδικα αλλά είναι και πιο ακριβής αυξάνοντας έτσι τελικά την ποιότητα του ελέγχου.

2.1.6 Αναβάθμιση και Εξέλιξη Έργου

Στη συνέχεια της αναζήτησής μας, μας απασχόλησε το πως οι προγραμματιστές αντιμετωπίζουν τον κώδικα που δημιουργούν και συγκεκριμένα τις αναβαθμίσεις του. Το άρθρο των Chatzimparmpas, A., Bibi, S., Zozas, I. and Kerren, A.[5], εξετάζει μέσω μιας ποιοτικής μελέτης σε ένα πλήθος 20 κορυφαίων έργων JavaScript, την ισχύ των νόμων εξέλιξης εφαρμογών, όπως αυτοί περιγράφηκαν από τον Lehman. Οι ερευνητές οργανώνουν τη μελέτη στην ανάλυση των 5 εκ των 9 νόμων. Συγκεκριμένα ξεκινώντας από τον νόμο της συνεχούς αλλαγής, σύμφωνα με τον οποίο: "ένα σύστημα πρέπει να προσαρμόζεται συνεχώς ειδάλλως γίνεται προσδευτικά λιγότερο ικανοποιητικό", παρατηρούν ότι ο νόμος εφαρμόζεται, χωρίς όμως να μπορεί να προσδιοριστεί επακριβώς το διάστημα αναβάθμισης ή το μέγεθος της αλλαγής. Αναφορικά με τον νόμο της πολυπλοκότητας, "καθώς ένα σύστημα εξελίσσεται, η πολυπλοκότητά του αυξάνεται εκτός εάν γίνει δουλειά για τη συντήρηση ή την ελάττωση της", όπως διαπιστώνουν οι ερευνητές η αύξηση πολυπλοκότητας αφορά την κατανόηση του κώδικα και δεν επηρεάζει την πολυπλοκότητα κατά τον έλεγχο και την εκτέλεση του. Όπως απέδειξαν, με το πέρας του χρόνου οι αναβάθμισεις των έργων μειώνονται παρότι η προσπάθεια που απαιτείται για την συντήρηση τους παραμένει αμετάβλητη. Χωρίς να μπορούν να επιβεβαιώσουν ή να απορρίψουν πλήρως τον νόμο Διατήρησης Σταθερότητας, μας παρέχουν γνώση ώστε να διαπιστώσουμε ότι υπάρχει κάποιο σημείο χρονικά που οι προγραμματιστές σταματούν να εξελίσσουν το έργο είτε επειδή οι προσθήκες σε κώδικα δεν προσφέρουν κάτι καινούργιο είτε διότι πλέον το έργο θεωρείται ξεπερασμένο. Στη συνέχεια μέσα από παρατήρηση κατέληξαν στην απόρριψη της θεώρησης ότι οι ραγδαίες αλλαγές προκαλούν προβλήματα στην απόδοση του κώδικα ενώ δεν επιβεβαιώσαν την ισχύ του νόμου που θέλει την ποιότητα του κώδικα να υποβαθμίζεται αναλογικά με τις αναβάθμισεις της εφαρμογής. Από την έρευνα συμπεραίνουμε ότι γίνεται συνεχής αναβάθμιση περιεχομένου εφαρμογής χωρίς δείγματα υποβάθμισης ποιότητας κώδικα ή αύξησης πολυπλοκότητας του.

Το άρθρο [12], πραγματεύεται την αναβάθμιση των βιβλιοθηκών και των εξαρτήσεων προς τρίτο λογισμικό σε σχέση με την εξέλιξη του κύριου λογισμικού. Η μελέτη αφορά ένα πλήθος 4.600 έργων λογισμικού εκ των οποίων μόνο το 18.5% διατηρεί ενημερωμένες τις εκδόσεις των τρίτων μερών λειτουργικού που απαιτούνται. Η ενημέρωση αφορά τα αρχεία που διατηρούν κατάλογο εξαρτήσεων προς τρίτο λογισμικό όπως τα package.json, σε περιπτώσεις έργων JavaScript. Στο άρθρο από την εισαγωγή οι συγγραφείς κάνουν ξεκάθαρο ότι ο κύριος λόγος για τον οποίο οι προγραμματιστές δεν κάνουν ενημέρωση απαιτήσεων είναι ο χρόνος που απαιτεί και η δυσκολία της διαδικασίας. Όπως τονίζουν, προτού οι προγραμματιστές προβούν σε αναβάθμιση βιβλιοθηκών εξετάζουν:

- α) τις νέες δυνατότητες που προστίθενται από την αναβάθμιση,
- β) την συμβατότητα με άλλα στοιχεία τρίτου λογισμικού σε σχέση με την αυτή της παρούσας έκδοσης που χρησιμοποιούν,
- γ) αν υπάρχουν αρκετά έργα στα οποία έχει γίνει αυτή η μετάβαση, άρα την αξιοπιστία της και τέλος
- δ) την τεχνική υποστήριξη που παρέχεται για την νέα έκδοση.

Η μελέτη είναι χωρισμένη σε τρία μέρη και εξετάζει σε τι βαθμό προχωρούν οι προγραμματιστές σε ενημέρωση εξαρτήσεων, πως αντιδρούν σε σημαντικές ανακοινώσεις νέων εκδόσεων και διωρθώσεων στην ασφάλεια και γιατί τις αγνοούν. Ως προς το πρώτο ερώτημα μέσα από μια ανάλυση Spearman-Pearson οι συγγραφείς χρησιμοποιούν το Correlation Coefficient ως δείκτη συσχέτισης πλήθους βιβλιοθηκών που χρησιμοποιούνται και αναβαθμίσεων εξαρτήσεων που πραγματοποιούνται. Με τον χαμηλό δείκτη να επιβεβαιώνει την υπόθεση ότι το πλήθος των εξαρτήσεων δεν επηρεάζει την συχνότητα αναβάθμισης. Όπως διαπιστώνουν οι συγγραφείς, σε έργα τα οποία εξαρτώνται από πολλά στοιχεία τρίτου λογισμικού οι προγραμματιστές επιλέγουν σπάνια να προβούν σε σημαντικές αλλαγές που θα επηρεάσουν μεγάλο κομμάτι της εφαρμογής και προτιμούν να παραμείνουν σε παλαιότερες σταθερές εκδόσεις βιβλιοθηκών. Αναφέρουν στη συνέχεια ότι ο λόγος για τον οποίο οι προγραμματιστές αποφεύγουν την αναβάθμιση είναι η αύξηση των εξαρτήσεων προς τρίτο λογισμικό που μπορεί να επιφέρει σε σχέση με τις εκδόσεις των βιβλιοθηκών που ήδη χρησιμοποιούνται για τις ανάγκες του έργου. Οι επιπρόσθετες εξαρτήσεις αυξάνουν την πολυπλοκότητα του κώδικα αλλά και της διαδικασίας συντήρησης. Μέσα από ένα ερωτηματολόγιο προς προγραμματιστές έργων μας γίνεται γνωστό ότι οι πρώτοι αντιμετωπίζουν την ενημέρωση των βιβλιοθηκών και των εξαρτήσεων ως χρονοβόρα διαδικασία χαμηλής προτεραιότητας, που συχνά προκαλεί μεγαλύτερα προβλήματα από τις δυνατότητες που προσφέρει. Καταλήγουν στο συμπέρασμα ότι ενώ οι προγραμματιστές κάνουν συχνές αναβαθμίσεις στον κώδικα που παράγουν δεν αναβαθμίζουν με την ίδια συχνότητα το τρίτο λογισμικό που χρησιμοποιούν.

Με την ολοκλήρωση της βιβλιογραφικής αναζήτησης καταλήξαμε στα θέματα που θέλουμε να θίξουμε μέσα από την μελέτη αυτή. Στο επόμενο κεφάλαιο γίνεται αναλυτική περιγραφή των ερωτημάτων και των μεθόδων που χρησιμοποιήθηκαν για την εξαγωγή συμπερασμάτων.

Κεφάλαιο 3: Ανάλυση και Σχεδίαση Θέματος

Σε αυτό το κεφάλαιο θα αναλύσουμε τον σκελετό της μελέτης, τα ερευνητικά ερωτήματα που μας απασχολούν και τον τρόπο που προσεγγίζτηκαν.

3.1 Στόχος και Ερευνητικά Ερωτήματα

Στόχο της μελέτης αποτελεί ο προσδιορισμός και η κατηγοριοποίηση των δημοφιλέστερων στοιχείων ανοιχτού κώδικα που επαναχρησιμοποιούνται μεμονωμένα ή σε ζεύγη από τους προγραμματιστές κατά την ανάπτυξη έργων λογισμικού JavaScript. Αυτός ο στόχος διατυπώνεται στα ακόλουθα ερευνητικά ερωτήματα:

[RQ1]: Ποια στοιχεία ανοιχτού κώδικα JavaScript επιλέγονται συχνότερα αναφορικά με την λειτουργικότητα που προσφέρουν;

Βασικός στόχος του ερωτήματος είναι ο εντοπισμός εξαιρετικά επαναχρησιμοποιήσιμων στοιχείων JavaScript και η καταγραφή της λειτουργικότητας που προσφέρουν. Μέσα από την μελέτη περίπτωσης, εντοπίζουμε τα στοιχεία που καλύπτουν ένα ποσοστό 70% των πακέτων που χρησιμοποιούνται στα 430 έργα. Στη συνέχεια αναλύουμε την πορεία λήψεων στο χρόνο για τα στοιχεία αυτά και προσδιορίζουμε τη σχέση μεταξύ δημοφιλίας και πλήθους απόκτησης. Για τα ίδια στοιχεία κάνουμε μελέτη εξαρτήσεων προς και από τρίτο λογισμικό. Συνολικά η ανάλυση θα παρέχει μια εικόνα των τύπων λειτουργιών που επαναχρησιμοποιούνται επί το πλείστον στο πλαίσιο της ανάπτυξης εφαρμογών JavaScript καθώς και τον λόγο για τον οποίο κάποια στοιχεία επιλέγονται πιο συχνά έναντι σε άλλα.

[RQ2]: Ποια είναι η επαναχρησιμοποίηση στοιχείων JavaScript ανοιχτού κώδικα σε σχέση με την έκδοσή τους;

Αυτή η ερώτηση εξετάζει τα επαναχρησιμοποιούμενα στοιχεία σε σχέση με την έκδοση που επιλέγεται περισσότερο. Με χρήση γραφήματος απεικονίζεται η πορεία μετάβασης σε διαφορετικές εκδόσεις και το ποσοστό χρήσης κάθε έκδοσης τρίτου λογισμικού. Έπειτα από τα αποτελέσματα της χρήσης κάθε έκδοσης καταγράφεται η πιο δημοφιλής και ο χαρακτηρισμός της σε σχέση με την πιο πρόσφατη έκδοση που κυκλοφορεί για το κάθε λογισμικό. Μέσω μετρικών και γραφικών παραστάσεων το ερώτημα αυτό συμβάλει στην εξαγωγή ενός μοτίβου επιλογής μεταξύ της παραμονής σε παλιότερη έκδοση και της μετάβασης στην πιο πρόσφατη έκδοση του λογισμικού τρίτου μέρους. Τέλος, γίνεται απαρίθμηση των πλεονεκτημάτων και μειονεκτημάτων που προσφέρει κάθε μία από τις δύο τακτικές ως προς την σταθερότητα της εφαρμογής αλλά και τις δυνατότητες αναβάθμισης και συντήρησης της.

[RQ3]: Ποια είναι η ένταση συνύπαρξης των δημοφιλών επαναχρησιμοποιήσιμων ζευγών στοιχείων ανοιχτού κώδικα JavaScript;

Το ερώτημα στοχεύει στον εντοπισμό των συνδυασμών στοιχείων JavaScript που χρησιμοποιούνται συχνά στο πλαίσιο ανάπτυξης μιας μεμονωμένης εφαρμογής. Μέσα από μια ανάλυση συσχέτισης θα λάβουμε μια εικόνα για τις δημοφιλέστερες επιλογές συνδυασμού λειτουργιών που προσφέρουν τα στοιχεία. Θα περιγράψουμε τα οφέλη της συνδυαστικής τεχνικής και θα καταγράψουμε τα προβλήματα που επιλύει ο συνδυασμός

λειτουργιών για τους προγραμματιστές. Επιπλέον από τα ζεύγη που παρουσιάζουν μεγαλύτερη συσχέτιση και βρίσκονται υψηλά στα επίπεδα δημοφιλίας θα εξαχθεί συμπέρασμα για την συμβατότητα που παρουσιάζουν τα διάφορα στοιχεία και την σταθερότητα που παρέχει η επιλογή του συνδυασμού τους στον προγραμματιστή.

3.2 Ανάλυση Μελέτης Περίπτωσης

Στο πλαίσιο της παρούσας μελέτης χρησιμοποιήθηκαν τα 430 πιο δημοφιλή, ως προς τα forks, έργα JavaScript που φιλοξενούνται στο αποθετήριο του GitHub, έως τον Φεβρουάριο του 2021. Η επιλογή αφορά εφαρμογές που πληρούν τα ακόλουθα κριτήρια:

- Έχουν διάρκεια ζωής τουλάχιστον δύο ετών
- Παρουσιάζουν περισσότερες από 5 κυκλοφορίες
- Έχουν τουλάχιστον μια αναβάθμιση στο περασμένο έτος που βρίσκεται στον κατάλογο tags και
- Έχουν κατ' ελάχιστον 1k αστέρια.

Παρότι ο αριθμός των έργων που μελετήθηκε είναι μικρός σε σχέση με το πλήθος των εφαρμογών JavaScript που υπάρχουν στα αποθετήρια κώδικα, θεωρούμε ότι αποτελεί ικανό δείγμα ώστε να παρέχει έμπιστα συμπεράσματα.

Για κάθε έργο έχει γίνει λήψη όλων των αρχείων που αποτελούν την τελευταία έκδοση του. Σε κάθε έργο έχουν χρησιμοποιηθεί αποκλειστικά τα αρχεία package.json, που παρέχουν πληροφορίες αναφορικά με τις εξαρτήσεις του έργου προς τρίτο λογισμικό. Οι εξαρτήσεις του έργου υποδεικνύουν τα στοιχεία που επαναχρησιμοποιούνται στο πλαίσιο ανάπτυξης εφαρμογών JavaScript. Συνολικά, εντοπίστηκαν 129 επαναχρησιμοποιούμενα στοιχεία.

3.2.1 Συγκέντρωση και Ανάλυση δεδομένων

Το πρώτο σύνολο μετρήσεων που χρησιμοποιούνται στο πεδίο αυτής της μελέτης πραγματοποιείται στο επίπεδο των συστατικών. Οι μετρήσεις προέρχονται από τα μεταδεδομένα που παρέχονται από τα αρχεία package.json, συμπεριλαμβανομένων των:

- Όνομα του πακέτου που επαναχρησιμοποιεί τρίτο λογισμικό,
- Όνομα και έκδοση του τρίτου λογισμικού που επαναχρησιμοποιείται
- Είδος εξάρτησης μεταξύ πακέτου και τρίτου λογισμικού

Για κάθε στοιχείο τρίτου μέρους, καταγράφηκε ο τύπος λειτουργικότητας που προσφέρει στην εφαρμογή. Καταλήγοντας σε ένα σύνολο 5 τύπων λειτουργιών:

• Μεταγλωττιστές (Compilers)

Ως Compilers, κατατάξαμε τα στοιχεία ανοιχτού λογισμικού που προσφέρουν την δυνατότητα εκτέλεσης κώδικα, διόρθωσης συντακτικών λαθών και οπτικοποίησης

αποτελέσματος. Σε αυτή την ομάδα προσθέσαμε τα στοιχεία που κάνουν συνένωση μεταξύ υπομονάδων λογισμικού για την παραγωγή της ενιαίας εφαρμογής.

• Πλαίσια Ανάπτυξης (Frameworks)

Μονάδες ανοιχτού λογισμικού που υποστηρίζουν την ανάπτυξη λογισμικού. Είναι σχεδιασμένα ώστε να υποστηρίζουν την αυτοματοποίηση κατά την παραγωγή κώδικα και ενθαρρύνουν την επαναχρησιμοποίηση λογισμικού. Ένα από τα πιο γνωστά frameworks JavaScript είναι το Angular.

• Μονάδες διασύνδεσιμότητας (Interoperability Units)

Τα στοιχεία ανοιχτού λογισμικού που προσδίδουν στην JavaScript ιδιότητες που ανήκουν σε άλλες γλώσσες προγραμματισμού, βιοηθούν στην πλήρη επικοινωνία μεταξύ των διαφορετικών μονάδων που αποτελούν την εφαρμογή και συμβάλλουν στην επίλυση προβλημάτων συμβατότητας στις πλατφόρμες που εκτελείται το λογισμικό, ανήκουν στην κατηγορία αυτή.

• Μονάδες Ελέγχου (Testing Units)

Μονάδες τρίτου λογισμικού που χρησιμοποιούνται κατά τον έλεγχο της αναπτυσσόμενης εφαρμογής JavaScript. Ακόμη ως μονάδες ελέγχου θεωρήθηκαν τα έτοιμα σενάρια που επαναχρησιμοποιούνται από τους προγραμματιστές όπως αναφέρονται στα package.json.

• Μονάδες Διεπαφής Χρήστη (User Interface Units)

Ως στοιχεία διεπαφής χρήστη έχουν χαρακτηριστεί οι μονάδες τρίτου λογισμικού που χρησιμοποιούνται από την εφαρμογή για την δημιουργία της διεπαφής. Επιπλέον θεωρώντας χρήστη και το άτομο που παράγει το λογισμικό, μονάδες λογισμικού που χρησιμοποιούνται στην αλλαγή της διεπαφής του προγραμματιστή ή για την διευκόλυνση του κατά την συγγραφή του έργου έχουν ενταχθεί σε αυτή την κατηγορία.

Σε δεύτερο επίπεδο έγιναν μετρήσεις που σχετίζονται με τη δημοτικότητα και την ένταση συσχέτισης των επαναχρησιμοποιούμενων στοιχείων, όπως προτείνεται από τους Kula et al., [9]. Σε αυτό το πλαίσιο υπολογίστηκαν τα ακόλουθα:

1. **UsedBy** υποδεικνύει το σύνολο των έργων JS $\{v_1, v_2, \dots, v_n\}$ που χρησιμοποιούν το στοιχείο u .

$$UsedBy(u) \equiv \{v \mid v \rightarrow u\} \quad (1)$$

2. **Popularity** ενός στοιχείου (u) αφορά τον αριθμό συσχετίσεων UsedBy στις οποίες συμμετέχει.

$$popularity(u) \equiv |UsedBy(u)| \quad (2)$$

3. **Popularity of coexistence pairs** υπολογίζεται για ζεύγη επαναχρησιμοποιούμενων στοιχείων (u, v) και υποδεικνύει τις φορές που το ζεύγος συνυπάρχει σε μία σχέση **UsedBy**.

$$\text{popularity}(u, v) \equiv | \text{UsedBy}(u) \cap \text{UsedBy}(v) | \quad (3)$$

4. **Intensity of coexistence pairs** αποτελεί την κανονικοποιημένη συχνότητα των πιο δημοφιλών ζευγών στοιχείων. Για διοθέν σύνολο επαναχρησιμοποιούμενων στοιχείων I με ζεύγη $x, y \in I$ υπολογίζεται η intensity όπως φαίνεται από τον τύπο:

$$\text{intensity}(x, y, I) = \frac{\text{popularity}(x, y)}{\max_{\substack{i, j \in I \\ i \neq j}} (\text{popularity}(i, j))} \quad (4)$$

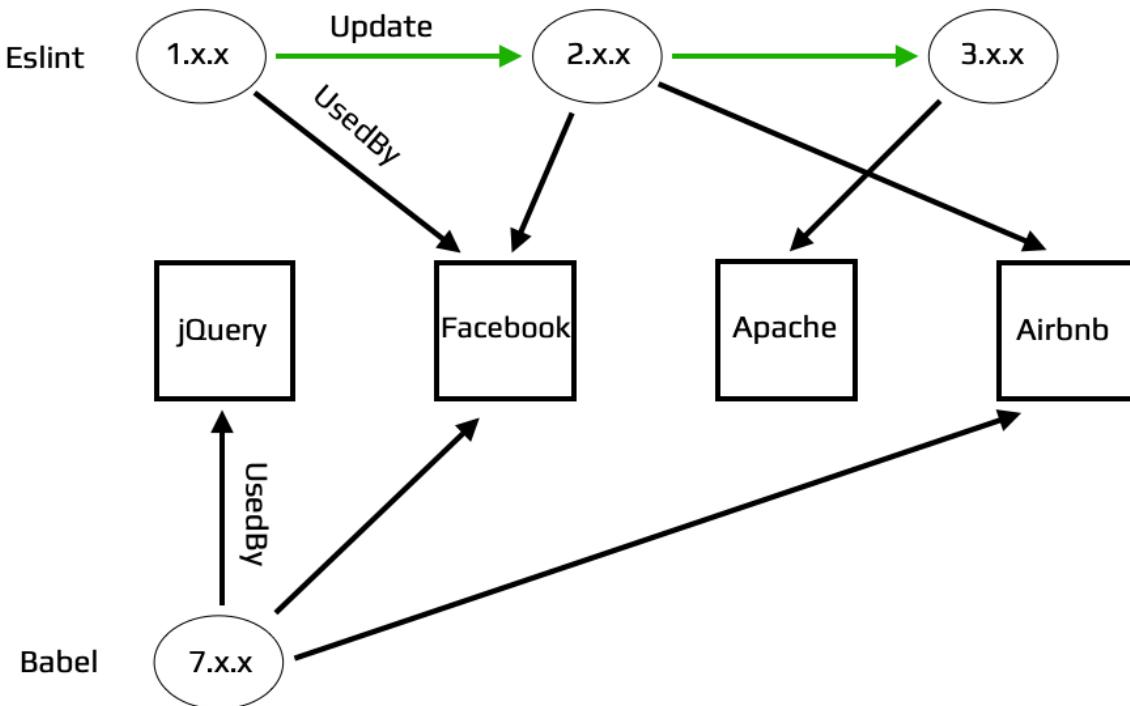
Όπου $x, y \in I$ και ταχ επιστρέφει τον μέγιστο αριθμό φορών επαναχρησιμοποίησης του ζεύγους στοιχείων.

Για να γίνει κατανοητή η διαδικασία των μετρήσεων αυτών θα παρουσιάσουμε ένα σύντομο παράδειγμα. Για χάρη παραδείγματος θα χρησιμοποιηθεί το στοιχείο babel και το στοιχείο eslint που αποτελούν και τα πιο δημοφιλή στοιχεία τρίτου λογισμικού μεταξύ των 129 που συναντώνται 9389 φορές στα 430 έργα που μελετήθηκαν, όπως θα δείξουμε στο RQ1.

Στο σχήμα, Εικόνα 1., τα τετράγωνα απεικονίζουν τα διαφορετικά έργα τα οποία χρησιμοποιούν τα στοιχεία τρίτου μέρους, eslint και babel, που εμφανίζονται με τα οιβάλ σχήματα. Με τα μαύρα βέλη έχουμε συμβολίσει τις σχέσεις **UsedBy**, ενώ τα πράσινα δείχνουν τις διαφορετικές εκδόσεις του ίδιου τρίτου μέρους που χρησιμοποιούνται.

Όπως αναφέρθηκε η σχέση **UsedBy** αφορά όλα τα project που χρησιμοποιούν τουλάχιστον μία φορά το εκάστοτε στοιχείο. Συνεπώς στο σχήμα το eslint σχηματίζει 3 σχέσεις **UsedBy** ενώ επαναχρησιμοποιείται συνολικά 4 φορές. Αντίστοιχα το babel σχηματίζει 3 σχέσεις **UsedBy** και χρησιμοποιείται συνολικά 3 φορές. Γίνεται αντιληπτό ότι η χρήση παραπάνω από μια φορές του ίδιου στοιχείου σε ένα έργο δε προσμετράται ως σχέση **UsedBy**. Επιπλέον το ποια έκδοση χρησιμοποιείται από κάθε έργο δεν επηρεάζει αυτές τις μετρήσεις απλά μας δίνει πληροφορία για την πιο δημοφιλή έκδοση κάθε πακέτου. Ακόμα, από το σχήμα μπορούμε να υπολογίσουμε την τιμή popularity of coexistence. Παρατηρούμε ότι τα δύο στοιχεία συμμετέχουν από κοινού σε 2 σχέσεις **UsedBy** άρα η τιμή αυτή είναι και το ζητούμενο.

Το γράφημα αυτό μπορεί να θεωρηθεί και γράφος εξαρτήσεων καθώς οι σχέσεις **UsedBy** μπορούν αντίστροφα να υποδηλώσουν τις εξαρτήσεις κάθε έργου προς τρίτο λογισμικό. Στο παράδειγμα μας το έργο Facebook εξαρτάται και από το babel και από το eslint ενώ το έργο jQuerry σχηματίζει σχέση εξάρτησης μόνο με το babel.



ΕΙΚΟΝΑ 1: ΠΑΡΑΔΕΙΓΜΑ ΓΡΑΦΟΥ ΕΠΑΝΑΧΡΗΣΙΜΟΠΟΙΗΣΗΣ ΠΑΚΕΤΩΝ ΑΠΟ ΕΡΓΑ ΛΟΓΙΣΜΙΚΟΥ .

Στα έργα JavaScript οι εξαρτήσεις προς τρίτο λογισμικό χωρίζονται σε 5 κατηγορίες:

- **Normal Dependencies** ή **Dependencies**:

Αφορούν τρίτο λογισμικό που απαιτείται από την εφαρμογή κατά την παραγωγή. Τα πακέτα που ανήκουν σε αυτή τη κατηγορία χρησιμεύουν ως βιβλιοθήκες για να λειτουργήσει ένα έργο.

- **Dev Dependencies**:

Αφορούν τρίτο λογισμικό που χρησιμεύει ως εργαλείο ανάπτυξης. Το τελικό προϊόν δεν χρειάζεται απαραίτητα αυτά τα πακέτα, τα οποία μπορεί να είναι:

- α) Εργαλεία μορφοποίησης, *Formatting tools* για την διόρθωση και τη μορφοποίηση το κώδικα εφαρμογής
- β) Μεταγλωττιστές κώδικα σε κώδικα ή *Transpilers*, είναι εργαλεία που διαβάζουν τον πηγαίο κώδικα γραμμένο σε μία γλώσσα προγραμματισμού και παράγουν τον ισοδύναμο κώδικα σε άλλη γλώσσα.
- γ) *Type definitions* ορίζουν τύπους μεταβλητών που αρχικά η JavaScript δεν υποστηρίζει
- δ) Ομαδοποιητές, *Bundlers* είναι εργαλεία που συγκεντρώνουν τον κώδικα και όλες τις εξαρτήσεις εφαρμογής σε ένα αρχείο JavaScript

ε) Βιβλιοθήκες με Σενάρια ελέγχου Test libraries, που προσφέρουν τα εργαλεία για τον έλεγχο του κώδικα εφαρμογής

στ) Document libraries παρέχουν βιβλιογραφία για τα πακέτα που χρησιμοποιούνται για τις ανάγκες της εφαρμογής ή γενική βιβλιογραφία για την συγγραφή κώδικα JavaScript.

• Peer Dependencies

Αυτά τα πακέτα απαιτούνται από ένα προϊόν για τη λειτουργία του, όπως συμβαίνει με τις κανονικές εξαρτήσεις. Ωστόσο, το προϊόν στο οποίο χρησιμοποιούνται δεν είναι τελικό αλλά είναι μια βιβλιοθήκη ή ένα plugin που χρησιμοποιείται από την τελική εφαρμογή.

• Optional Dependencies

Αφορούν τρίτο λογισμικό που χρησιμοποιείται από την εφαρμογή κατά την παραγωγή αλλά δεν είναι απαραίτητες και το προϊόν μπορεί να δημιουργηθεί και να εκτελεστεί χωρίς αυτές.

• Bundled Dependencies

Οι ομαδοποιημένες εξαρτήσεις αναφέρονται στη χρήση:

- α) πακέτου ή βιβλιοθήκης που δε προέρχεται από το πρωτότυπο.
- β) τροποποιημένου τρίτου λογισμικού που προέρχεται από το πρωτότυπο και
- γ) άλλου έργου που παράχθηκε από τον ίδιο τον προγραμματιστή ως λογισμικό τρίτου μέρους

Παρόλο που υπάρχουν 5 τύποι εξαρτήσεων οι προγραμματιστές σπάνια χρησιμοποιούν τους τύπους Optional Dependencies, Bundled Dependencies και Peer Dependencies καθώς οι πρώτες δεν είναι απαραίτητες, οι δεύτερες μπορούν ενσωματωθούν στις κανονικές εξαρτήσεις, dependencies ενώ οι τρίτες θεωρούνται υποκατηγορία των devDependencies. Συνεπώς, στο πλαίσιο της παρούσας μελέτης οι τύποι εξαρτήσεων τους οποίους θα συναντήσουμε είναι οι dependencies και dev Dependencies. Συνολικά από τις 9389 εξαρτήσεις που μελετήσαμε στα 430 έργα, οι 7182 αναφέρονται σε εξαρτήσεις dev Dependencies και μόνο οι 2207 αναφέρονται σε εξαρτήσεις dependencies. Στο Κεφάλαιο 5 παρουσιάζουμε την κατανομή των εξαρτήσεων και ερμηνεύουμε κατάλληλα αυτό το εύρημα.

3.2.2 Μέθοδοι Ανάλυσης

Για την εξαγωγή της έρευνας χρησιμοποιήθηκαν script και ειδικό λογισμικό για στατιστικές μελέτες το: Spss Statistics της IBM. Μετά την προεπεξεργασία δεδομένων, όπως περιγράφεται στην Ενότητα 4, έγινε η εισαγωγή τους στο Spss.

Για τις ανάγκες του ερωτήματος RQ1, χρησιμοποιήθηκαν περιγραφικά στατιστικά και συγκεκριμένα διαδικασίες συχνοτυκής ανάλυσης. Μέσω αυτής, βρέθηκαν τα πιο συχνά στοιχεία επαναχρησιμοποιούμενου λογισμικού, υπολογίστηκε το ποσοστό επανεμφάνισης ανά μονάδα και έγινε κατηγοριοποίηση ανά τύπο λειτουργικότητας. Σε αυτό το ερώτημα χρησιμοποιήθηκε το πλήθος των στοιχείων που καλύπτει το 70% της συνολικής διακύμανσης πακέτων ή αλλιώς τα 35 πιο συχνά χρησιμοποιούμενα στοιχεία τρίτου λογισμικού. Για την απεικόνιση των αποτελεσμάτων δημιουργήθηκε πίνακας που αναφέρει το όνομα επαναχρησιμοποιούμενου λογισμικού, το πλήθος των σχέσεων UsedBy, τις συνολικές εμφανίσεις σε όλα τα έργα, και τέλος την κατηγορία λειτουργικότητας κάθε στοιχείου. Επιπλέον με χρήση τρίτου λογισμικού προέκυψε η καμπύλη μεταβολής λήψεων του κάθε πακέτου σε ένα διάστημα πενταετίας. Ακόμα σε αυτό το ερώτημα από το GitHub και τα στοιχεία που παρέχουν οι προγραμματιστές έγινε καταγραφή των συνολικών εξαρτήσεων από και προς τρίτο λογισμικό, για καθένα από τα 35 πιο δημοφιλή στοιχεία. Τέλος πραγματοποιήθηκε μελέτη κατανομής είδους εξάρτησης που έχουν τα 430 έργα προς τα 35 πιο δημοφιλή πακέτα.

Στη συνέχεια για το ερώτημα RQ2 δημιουργήθηκε γράφημα στοιβαγμένων ράβδων που απεικονίζει τα ίδια 35 πιο δημοφιλή επαναχρησιμοποιούμενα στοιχεία, και την κατανομή των εκδόσεων που χρησιμοποιούνται από κάθε στοιχείο. Στο γράφημα χρησιμοποιούνται διαβαθμισμένα χρώματα για να υποδείξουμε τις διάφορες εκδόσεις λογισμικού. Για την δημιουργία του γραφήματος πραγματοποιήθηκε η διαδικασία cross tabulation, στα πεδία Όνομα λογισμικού τρίτου μέρους και Έκδοση λογισμικού τρίτου μέρους, στο Spss. Η διαδικασία αφορά την δημιουργία ενός νέου πίνακα συνδυάζοντας ιδιότητες από έναν αρχικό πίνακα. Αποτελεί μια μέθοδο ποσοτικής ανάλυσης της σχέσης μεταξύ πολλαπλών μεταβλητών. Για το ερευνητικό ερώτημα εξετάζουμε την σχέση μεταξύ λογισμικού και έκδοσης και δημιουργούμε έναν πίνακα με στήλες τις εκδόσεις λογισμικού και γραμμές το όνομα του λογισμικού τρίτου μέρους. Ως αποτέλεσμα λαμβάνουμε το ποσοστό χρήσης κάθε έκδοσης λογισμικού καθώς και την πιο δημοφιλή έκδοση λογισμικού για κάθε στοιχείο. Οι πληροφορίες του πίνακα τελικά μετατρέπονται σε γράφημα το οποίο αναλύεται στο Κεφάλαιο 6

Για το ερώτημα RQ3, για τον υπολογισμό της ένταση συνύπαρξης των δημοφιλών ζευγών εκτελέστηκε ανάλυση Spearman μέσω του λογισμικού Spss. Για να πραγματοποιήσουμε την ανάλυση Spearman χρειάστηκε να καταφύγουμε στη χρήση πίνακα crosstab μεταξύ των Όνομα λογισμικού τρίτου μέρους και Όνομα έργου JavaScript. Η ανάλυση Spearman εξετάζει την συσχέτιση μεταξύ δύο στοιχείων που λαμβάνουν διακριτές τιμές και δεν παρουσιάζουν γραμμική κατανομή. Στην ανάλυση Spearman τα αποτελέσματα που χρησιμοποιούνται για να χαρακτηριστεί η σχέση μεταξύ των ζευγών είναι:

- To Correlation Coefficient (Συντελεστής Συσχέτισης)

Αυτό το στοιχείο μας υποδεικνύει την ένταση της συσχέτισης μεταξύ των ζευγών. Οι τιμές που μπορεί να λάβει αυτή η μετρική είναι από -1 έως 1, με την πρώτη να είναι η

ισχυρότερη αρνητική και η δεύτερη να είναι ισχυρότερη θετική. Για τις ενδιάμεσες τιμές θεωρούμε:

- [>0.5] ισχυρή θετική
- [0.3-0.49] μέτρια θετική
- [0.1-0.29] αδύναμη θετική

Με τον ίδιο τρόπο χαρακτηρίζουμε και τις αντίστοιχες αρνητικές

• Το Significance (Σημαντικότητα)

Αυτό το στοιχείο ορίζει την σημαντικότητα της συσχέτισης σε σχέση με τον συντελεστή α που έχουμε ορίσει. Θεωρούμε ότι $\alpha=0.01$ και έχουμε χαρακτηρίσει την σημαντικότητα των συσχετίσεων ως:

- [<0.01] υψηλά σημαντική
- [0.01-0.05] σημαντική
- [>0.05] χαμηλά σημαντική

Συνεπώς συνδυάζοντας τις δύο μετρικές χαρακτηρίζουμε τις σχέσεις πρώτα ώς προς την ένταση και έπειτα ώς προς την σημαντικότητα της συσχέτισης. Κατά την παραγωγή των αποτελεσμάτων το SPSS χρησιμοποιώντας αστερίσκους τονίζει ποιες σχέσεις είναι σημαντικές και πρέπει να προσέξει ο χρήστης.

Στο επόμενο βήμα της ανάλυσης μας, δημιουργήσαμε έναν πίνακα ζευγών στοιχείων και έργων JavaScirpt ώστε να υπολογίσουμε το Popularity of coexistence pairs. Το άθροισμα κάθε στήλης μας δίνει την τιμή Popularity of coexistence pairs ενώ το άθροισμα κάθε γραμμής μας δίνει πληροφορίες για τον αριθμό των ζευγών που χρησιμοποιεί κάθε project. Έπειτα από τα αποτελέσματα της ανάλυσης αυτής χρησιμοποιώντας τον τύπο (4) υπολογίσαμε η ένταση συνύπαρξης των ζευγών και κατασκευάσαμε πίνακα που απεικονίζει την αριθμητική τιμή της έντασης. Η ένταση λαμβάνει τιμές από 0 έως 1. Για να παρουσιάσουμε την διακύμανση της έντασης, να εντοπίσουμε τα ζεύγη με την μεγαλύτερη τιμή και να τα χαρακτηρίσουμε ως προς την λειτουργικότητα τους χρησιμοποιήσαμε χρωματισμό ανάλογο με την αριθμητική τιμή. Ξεκινώντας από το πράσινο για τιμές κοντά στο 0 και καταλήγοντας στο κόκκινο για τιμές κοντά στο 1 παράχθηκε ο χάρτης θερμότητας που παρουσιάζουμε στην Ενότητα 6.

Πίνακας 1: Απεικόνισης Χρήσης Εργαλείων-Μεθόδων ανά Ερευνητικό Ερώτημα.

Ερώτημα	Μεταβλητές	Μέθοδοι
RQ1	Όνομα λογισμικού τρίτου μέρους Έιδος εξάρτησης προς τρίτο λογισμικό	Frequency analysis Πίνακας ανάλυσης πλήθους εξαρτήσεων ανά είδος και ανά πακέτο
RQ2	Όνομα λογισμικού τρίτου μέρους Έκδοση λογισμικού τρίτου μέρους	Cross tabulation Γράφημα στοιβαγμένων στηλών

Ερώτημα	Μεταβλητές	Μέθοδοι
RQ3	'Όνομα λογισμικού τρίτου μέρους 'Όνομα έργου JavaScript	Cross tabulation Spearman analysis Υπολογισμός Popularity of coexistence pairs Υπολογισμός Intensity

Πίνακας 2: Απεικόνισης Χρήσης Μεταβλητών ανά Μέθοδο και Ερευνητικό Ερώτημα.

Ερώτημα	Εργαλείο	Μέθοδοι
RQ1	Spss	Frequency analysis Δημιουργία πίνακα Συχνοτήτων και λειτουργιών
RQ2	Spss	Crosstab πίνακας μεταξύ των 'Όνομα Λογισμικού Τρίτου Μέρους και 'Έκδοσης Λογισμικού Τρίτου Μέρους Εξαγωγή Γραφήματος
RQ3	Spss	Spearman Ανάλυση Υπολογισμός Popularity of coexistence pairs Υπολογισμός Intensity Χάρτης Θερμότητας

Πίνακας 3: Απεικόνισης Χρήσης Μεταβλητών ανά Σχέση .

Μεταβλητές	Σχέση
'Όνομα λογισμικού τρίτου μέρους 'Όνομα έργου JavaScript	UsedBy Popularity Popularity of coexistence pairs Intensity of coexistence pairs

Ο λόγος που χρησιμοποιείται μόνο αυτό το ζεύγος μεταβλητών είναι η εξάρτηση των τεσσάρων σχέσεων. Η UsedBy χρησιμοποιεί άμεσα τα δεδομένα 'Όνομα λογισμικού τρίτου μέρους και 'Όνομα έργου JavaScript. Η ίδια παράγει την popularity και την Popularity of coexistence pairs συνεπώς οι δύο αυτές σχέσεις εξαρτώνται έμεσσα από τις μεταβλητές 'Όνομα λογισμικού τρίτου μέρους και 'Όνομα έργου JavaScript. Τέλος η Intensity of coexistence pairs χρησιμοποιεί ως δεδομένα τα αποτελεσματα των προηγούμενων σχέσεων.

Συνολικά για την έρευνα μας τα δεδομένα που χρησιμοποιήσαμε από τα έργα είναι:

- **Όνομα Έργου JavaScript**
- **Έκδοση Έργου JavaScript**
- **Όνομα τρίτου λογισμικού**
- **Έκδοση τρίτου λογισμικού**
- **Είδος εξάρτησης προς τρίτο λογισμικό**

Κεφάλαιο 4: Υλοποίηση

Έχοντας περιγράψει γενικά τον τρόπο με τον οποίο παράχθηκαν τα αποτελέσματα της μελέτης, σε αυτό το κεφάλαιο θα αναλύσουμε τον τρόπο προεπεξεργασίας δεδομένων, τις παραδοχές που έγιναν και θα περιγράψουμε τα εργαλεία που δημιουργήθηκαν για να ικανοποιήσουν τις ανάγκες της μελέτης.

4.1 Λεπτομέρειες υλοποίησης

Για την υλοποίηση της εργασίας χρησιμοποιήθηκαν αποκλειστικά τα αρχεία package.json. Τα αρχεία package.json συνοδεύουν κάθε έργο JavaScript. Το έργο μπορεί να έχει πολλαπλά package.json ανάλογα με το πλήθος πακέτων που χρησιμοποιούνται κατά την ανάπτυξη του. Τα αρχεία αυτά δίνουν πληροφορίες σχετικά με¹:

- Το όνομα του πακέτου
- Την έκδοση του πακέτου
- Μια σύντομη περιγραφή της λειτουργίας του
- Τις λέξεις κλειδιά για την αναζήτησή του
- Την άδεια (licence)
- Την πηγή προέλευσης του
- Τα bugs που υπάρχουν
- Μια σύντομη περιγραφή της λειτουργίας του
- Τα αρχεία που χρησιμοποιεί
- Τις εξαρτήσεις προς άλλα πακέτα ή βιβλιοθήκες που υπάρχουν
- Την έκδοση του λογισμικού εξάρτησης και
- Το είδος των εξαρτήσεων προς τρίτο λογισμικό

Στο πλαίσιο της διπλωματικής οι πληροφορίες που μελετήθηκαν είναι οι εξαρτήσεις προς τρίτο λογισμικό και συγκεκριμένα η επιλογή των εκδόσεων τρίτου λογισμικού. Ακολούθως υπάρχει εικόνα του αρχείου json που περιγράφηκε.

¹ <https://docs.npmjs.com/cli/v7/configuring-npm/package-json>

```

1  {
2   "name": "chart.js",
3   "homepage": "https://www.chartjs.org",
4   "description": "Simple HTML5 charts using the canvas element.",
5   "version": "2.9.4",
6   "license": "MIT",
7   "jsdelivr": "dist/Chart.min.js",
8   "unpkg": "dist/Chart.min.js",
9   "main": "dist/Chart.js",
10  "keywords": [
11    "canvas",
12    "charts",
13    "data",
14    "graphs",
15    "html5",
16    "responsive"
17  ],
18  "repository": {
19    "type": "git",
20    "url": "https://github.com/chartjs/Chart.js.git"
21  },
22  "bugs": {
23    "url": "https://github.com/chartjs/Chart.js/issues"
24  },
25  "files": [
26    "bower.json",
27    "composer.json",
28    "dist/*.css",
29    "dist/*.js"
30  ],
31  "devDependencies": {
32    "clean-css": "^4.2.1",
33    "rollup-plugin-terser": "^5.0.0",
34    "yargs": "^12.0.5"
35  },
36  "dependencies": {
37    "chartjs-color": "^2.1.0",
38    "moment": "^2.10.2"
39  }
40}

```

ΕΙΚΟΝΑ 2: ΠΑΡΑΔΕΙΓΜΑ ΑΡΧΕΙΟΥ PACKAGE.JSON.

Για τις ανάγκες της εργασίας, δημιουργήθηκαν δύο bash script τα οποία χρησιμοποιήθηκαν για να αυτοματοποιηθεί η διαδικασία λήψης των έργων JavaScript από το GitHub και η προεπεξεργασία των δεδομένων τους.

Το πρώτο script τρέχει για κάθε έργο μεμονωμένα ενώ το δεύτερο εκτελείται για να παραχθεί το τελικό csv που χρησιμοποιείται για την μελέτη.

Η δομή που ακολουθεί το πρώτο script είναι:

1. Λήψη έργου

Το script δέχεται δύο ορίσματα το όνομα του συγγραφέα-προφίλ και το όνομα του repository του έργου. Με την χρήση της εντολής wget και κατάλληλες παραμέτρους πραγματοποιείται λήψη μόνο του zip αρχείου της τελευταίας πιο πρόσφατης έκδοσης του λογισμικού από την κατηγορία tags του GitHub.

2. Αποσυμπίεση και αποθήκευση φακέλων σε νέο κατάλογο (directory)

Στη συνέχεια με την εντολή unzip γίνεται αποσυμπίεση των αρχείων σε φάκελο που έχει την ίδια ονομασία με το έργο.

3. Διαχωρισμός και διαγραφή των υποκαταλόγων που δεν περιέχουν αρχεία package.json και των υπολοίπων αρχείων

Η διαδικασία γίνεται με την εντολή find. -not και find. -type d -empty.

4. Επεξεργασία των αρχείων package.json

Σε αυτό το βήμα από τα αρχεία package.json διαγράφονται ειδικοί χαρακτήρες, και κενές γραμμές ώστε να γίνει ευκολότερη η εξαγωγή των στοιχείων που ενδιαφέρουν την μελέτη. Για να γίνει κατανοητό από τον αναγνώστη έχει γίνει παράθεση στην προηγούμενη ενότητα εικόνα του αρχικού package.json και ακολούθως υπάρχει εικόνα του αρχείου μετά την επεξεργασία.

```
package2.json > No Selection
1  name:chart.js
2  homepage:https://www.chartjs.org
3  description:SimpleHTML5chartsusingthecanvaselement.
4  version:2.9.4
5  license:MIT
6  jsdelivr:dist/Chart.min.js
7  unpkg:dist/Chart.min.js
8  main:dist/Chart.js
9  keywords:
10 canvas
11 charts
12 data
13 graphs
14 html5
15 responsive
16
17 repository:
18 type:git
19 url:https://github.com/chartjs/Chart.js.git
20 }
21 bugs:
22 url:https://github.com/chartjs/Chart.js/issues
23 }
24 files:
25 bower.json
26 composer.json
27 dist/*.css
28 dist/*.js
29
30 devDependencies:
31 clean-css:^4.2.1
32 rollup-plugin-terser:^5.0.0
33 args:^12.0.5
34 }
35 dependencies:
36 chartjs-color:^2.1.0
37 moment:^2.10.2
38 }
39 }
```

ΕΙΚΟΝΑ 3: ΠΑΡΑΔΕΙΓΜΑ ΑΡΧΕΙΟΥ PACKAGE.JSON ΜΕΤΑ ΤΗΝ ΕΠΕΞΕΡΓΑΣΙΑ.

5. Δημιουργία csv για το εκάστοτε έργο

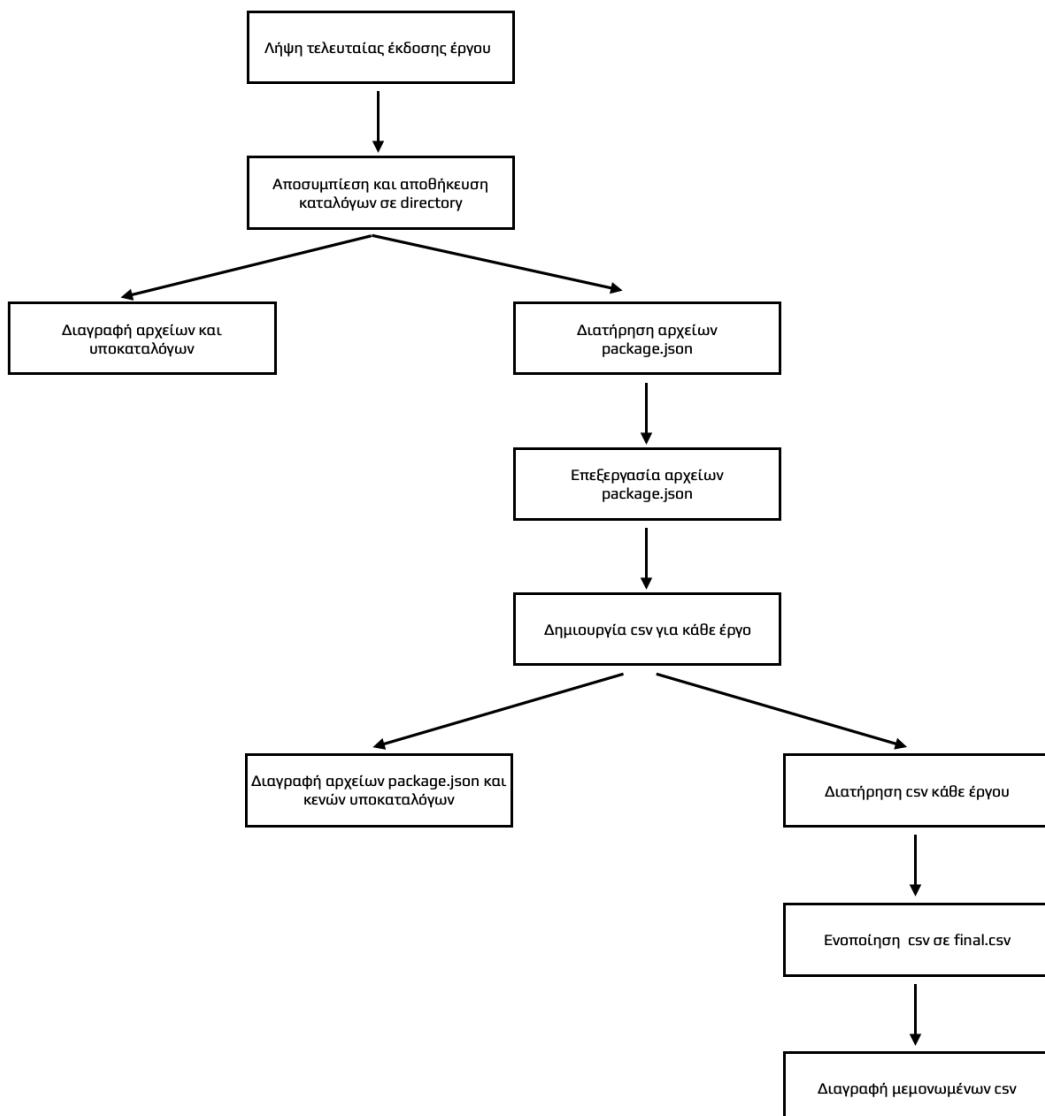
Κάθε αρχείο csv που παράγεται περιέχει από τα package.json πληροφορίες που αφορούν το όνομα του πακέτου, το είδος εξάρτησης, το όνομα του λογισμικού τρίτου μέρους και την έκδοση που χρησιμοποιείται.

6. Διαγραφή κενών υποκαταλόγων και όλων των αρχείων εκτός του csv

Η δομή που ακολουθεί το δεύτερο script είναι:

1. Συνένωση όλων των csv σε ένα τελικό

2. Διαγραφή των μεμονωμένων csv



ΕΙΚΟΝΑ 4: ΔΙΑΓΡΑΜΜΑΤΙΚΗ ΑΠΕΙΚΟΝΙΣΗ ΔΙΑΔΙΚΑΣΙΑΣ SCRIPT.

Έχουν συμπεριληφθεί στα Παραρτήματα στιγμιότυπα από τα bash script και από το τελικό csv.

Σε αυτό το στάδιο χρησιμοποιώντας πρόγραμμα διαχείρισης υπολογιστών φύλλων, έγινε η τελική επεξεργασία του csv. Για τις ανάγκες της μελέτης, ώστε να γίνει μια ολοκληρωμένη παρουσίαση των εκδόσεων που χρησιμοποιήθηκαν από κάθε λογισμικό τρίτου μέρους, έχει πραγματοποιηθεί απλοποίηση και συνένωση των υποεκδοσεων στον αριθμό της έκδοσης. Συνεπώς η έκδοση 9.0.1 και η 9.9.8 θα αναφέρονται ως 9.x.x. Με τον τρόπο αυτό τα αποτελέσματα είναι πιο ευανάγνωστα και κατανοητά στον αναγνώστη. Η γενίκευση των εκδόσεων δίνει μια ευρεία εικόνα και όχι το αποτέλεσμα μιας εξειδικευμένης περίπτωσης, όπως θα γινόταν με την διατήρηση των υποεκδόσεων. Παρατηρήθηκε ότι σε αρχεία package.json που χρησιμοποιούσαν το ίδιο τρίτο λογισμικό το όνομα δεν ήταν γραμμένο με μία τυποποιημένη μορφή. Με σκοπό να επιλυθεί αυτό το πρόβλημα και να αποφευχθεί η ύπαρξη διπλότυπων αναφορών στο λογισμικό υπό άλλο τίτλο, έγινε μετονομασία στο πιο επανεμφανιζόμενο όνομα για το εκάστοτε λογισμικό. Τέλος πακέτα με ελλείπεις πληροφορίες ή πακέτα που εμφανίζονταν λιγότερες από 15 φορές δε συμπεριλήφθηκαν στη μελέτη και διαγράφηκαν.

Κεφάλαιο 5: Αποτελέσματα

Το κεφάλαιο αυτό είναι αφιερωμένο στην απάντηση των ερευνητικών ερωτημάτων της μελέτης. Τα αποτελέσματα παρουσιάζονται μέσα από γραφήματα και στη συνέχεια ερμηνεύονται κατάλληλα.

5.1 Αποτελέσματα

5.1.1 Ερευνητικό ερώτημα 1

[RQ1]: Ποια στοιχεία ανοιχτού κώδικα JavaScript επιλέγονται συχνότερα αναφορικά με την λειτουργικότητα που προσφέρουν;

Πίνακας 4: Απεικόνιση Δημοφιλέστερων Στοιχείων JavaScript στα 430 έργα

Όνομα Τρίτου Λογισμικού	Popularity	Συνολικές εμφανίσεις σε όλα τα έργα	Λειτουργικότητα
Eslint	258	978	Testing Unit
Babel	216	1157	Compiler
Mocha	194	194	Testing Unit
Webpack	140	339	Interoperability Unit
Chai	117	117	Framework
TypeScript	108	176	Interoperability Unit
Karma	106	599	Testing Unit
Sinon	105	129	Testing Unit
Prettier	104	105	Compiler
Grunt	102	551	Interoperability Unit
Rollup	102	419	Interoperability Unit
Rimraf	101	101	Interoperability Unit
Lodash	99	99	Compiler
Express	86	87	Framework
Gulp	86	324	Compiler
Semver	84	84	Compiler
Glob	81	81	Interoperability Unit
Husky	79	79	Interoperability Unit
Nyc	77	77	Interoperability Unit
Browserify	72	72	Interoperability Unit
Chalk	72	73	Interoperability Unit
React	71	171	User Interface
Cross-env	70	70	Framework
Jest	68	69	Testing Unit
Fs-extra	67	68	Interoperability Unit
Jquery	63	63	User Interface

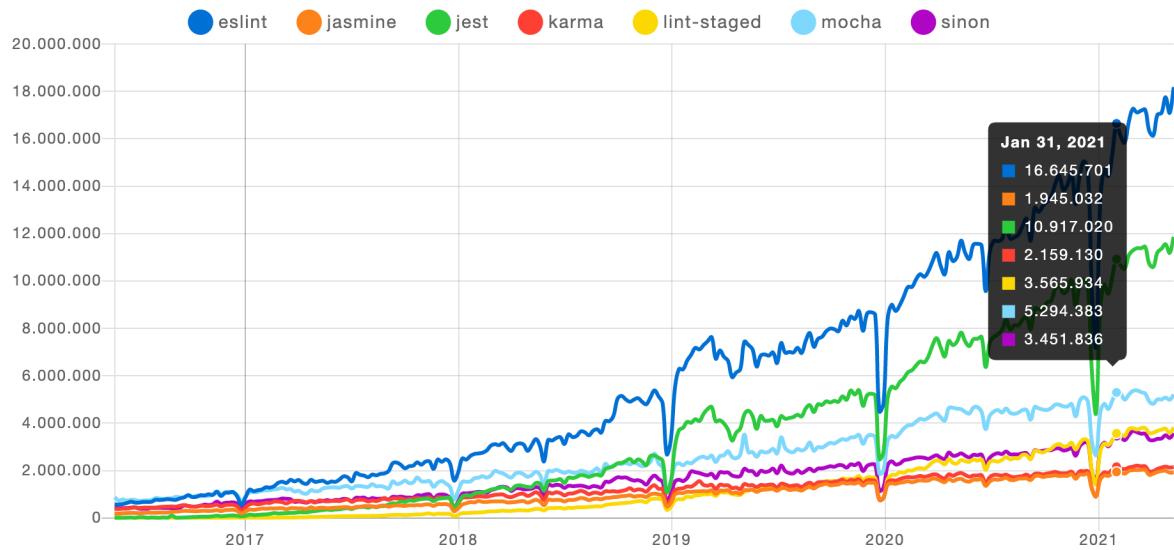
Όνομα Τρίτου Λογισμικού	Popularity	Συνολικές εμφανίσεις σε όλα τα έργα	Λειτουργικότητα
Uglify-js	62	62	Interoperability Unit
Lint-staged	61	61	Testing Unit
Coveralls	58	59	Testing Unit
Core-js	57	57	Framework
Css-loader	57	57	User Interface
Postcss	49	60	User Interface
Jasmine	45	57	Testing Unit
Sass	45	57	User Interface
Vue	27	66	User Interface

Ο Πίνακας 4., παρουσιάζει τα πιο δημοφιλή επαναχρησιμοποιούμενα στοιχεία JavaScript, καθώς και την κατηγορία που κατατάχθηκαν σύμφωνα με την λειτουργικότητα που προσφέρουν. Τα στοιχεία τρίτου λογισμικού αναλύουν πάνω από το 70% της συνολικής διακύμανσης. Από την συχνότητα επανεμφάνισης παρατηρείται ότι τα Testing Units, ακολουθούμενα από τους Compilers και τα Interoperability Units κυριαρχούν στην επιλογή των προγραμματιστών.

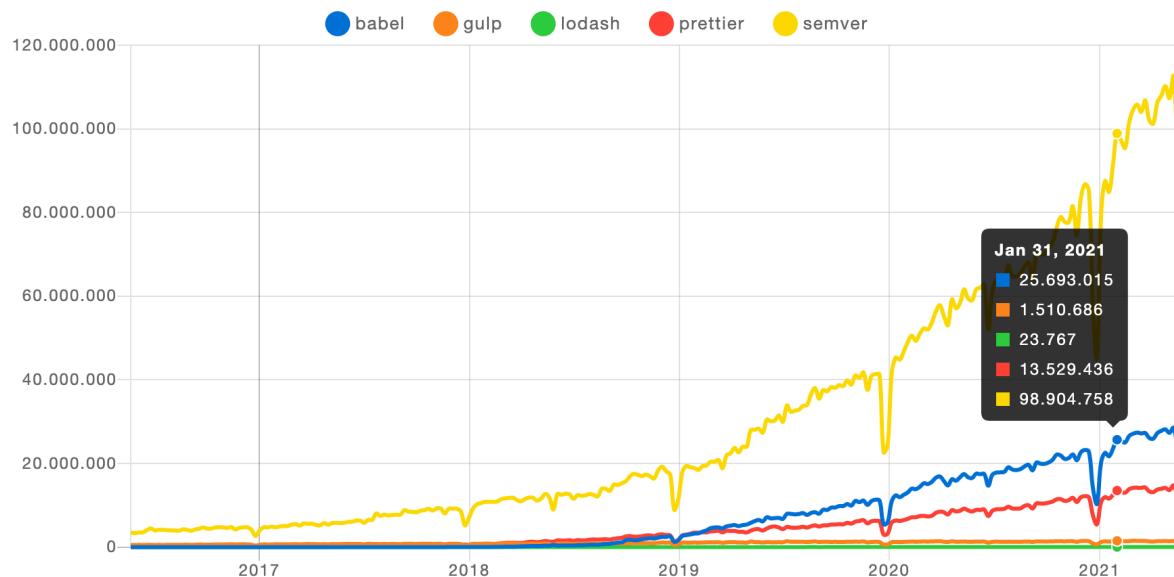
Θα αναλύσουμε σύντομα την λειτουργία των 5 πιο δημοφιλών πακέτων:

- **Eslint:** Αποτελεί ένα εργαλείο στατικής ανάλυσης κώδικα για τον εντοπισμό προβληματικών μοτίβων που βρίσκονται στον κώδικα JavaScript. Οι κανόνες του Eslint είναι διαμορφώσιμοι και ο χρήστης μπορεί να ορίσει και να χρησιμοποιήσει εξατομικευμένους κανόνες. Το Eslint επιλύει προβλήματα ποιότητας κώδικα και συγγραφής κώδικα.
- **Babel:** Χρησιμοποιείται κυρίως για τη μετατροπή του κώδικα ECMAScript 2015+ (ES6+) σε μια συμβατή προς τα πίσω έκδοση JavaScript που μπορεί να εκτελεστεί σε όλους τους browsers. Οι προγραμματιστές μπορούν να χρησιμοποιήσουν νέες δυνατότητες γλώσσας JavaScript και στη συνέχεια, χρησιμοποιώντας το Babel για να μετατρέψουν τον πηγαίο κώδικα σε εκδόσεις JavaScript που μπορούν να επεξεργαστούν τα προγράμματα περιήγησης στο Web.
- **Mocha:** Είναι ένα testing framework JavaScript για προγράμματα Node.js, με υποστήριξη προγράμματος περιήγησης, ασύγχρονες δοκιμές, αναφορές κάλυψης δοκιμών και χρήση οποιασδήποτε βιβλιοθήκης ισχυρισμών, assertion.
- **Webpack:** Αναγνωρίζει τα διαφορετικά στοιχεία που χρησιμοποιούνται για την παραγωγή εφαρμογής, διαχωρίζει και ομαδοποιεί τις εξαρτήσεις και παράγει ένα ενιαίο εκτελέσιμο. Το Webpack παίρνει τις εξαρτήσεις και δημιουργεί ένα γράφημα εξάρτησης που επιτρέπει στους προγραμματιστές ίστοι να χρησιμοποιούν μια αρθρωτή προσέγγιση για τους σκοπούς ανάπτυξης εφαρμογών ίστοι.
- **Chai:** Το Chai είναι μια βιβλιοθήκη ισχυρισμών, assertion BDD / TDD για nodes και για το πρόγραμμα περιήγησης που μπορεί να συνδυαστεί με οποιοδήποτε testing framework JavaScript.

Χρησιμοποιώντας τρίτο λογισμικό² για τα 35 πιο δημοφιλή στοιχεία όπως περιγράφηκαν παραπάνω πραγματοποιήθηκε σύγκριση ως προς τον αριθμό των λήψεων ανά στοιχείο και κατηγορία. Στα στιγμιότυπα που ακολουθούν φαίνεται η καμπύλη πορείας των λήψεων σε ένα διάστημα 5 ετών καθώς και το πλήθος των συνολικών λήψεων μέχρι και τον Ιανουάριο του 2021. Επιλέξαμε να γίνει η ανάλυση μέχρι εκείνο το χρονικό σημείο καθώς η λήψη των έργων πραγματοποιήθηκε τον Φεβρουάριο του 2021.

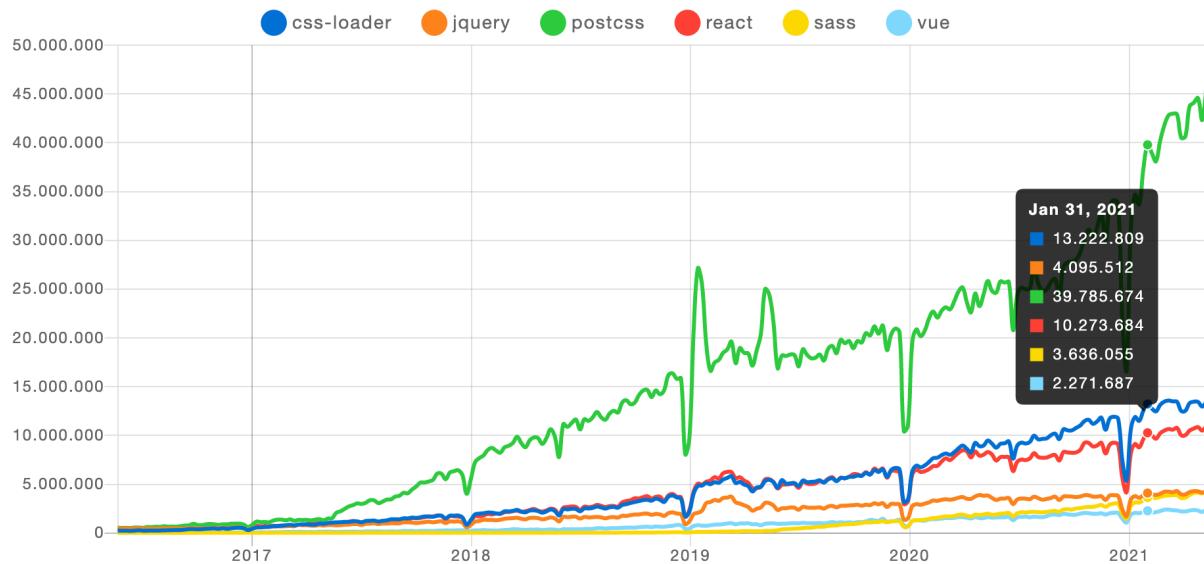


ΕΙΚΟΝΑ 5: ΚΑΜΠΥΛΗ ΛΗΨΗΣ TESTING UNITS ΔΙΑΣΤΗΜΑ 5 ΕΤΩΝ.

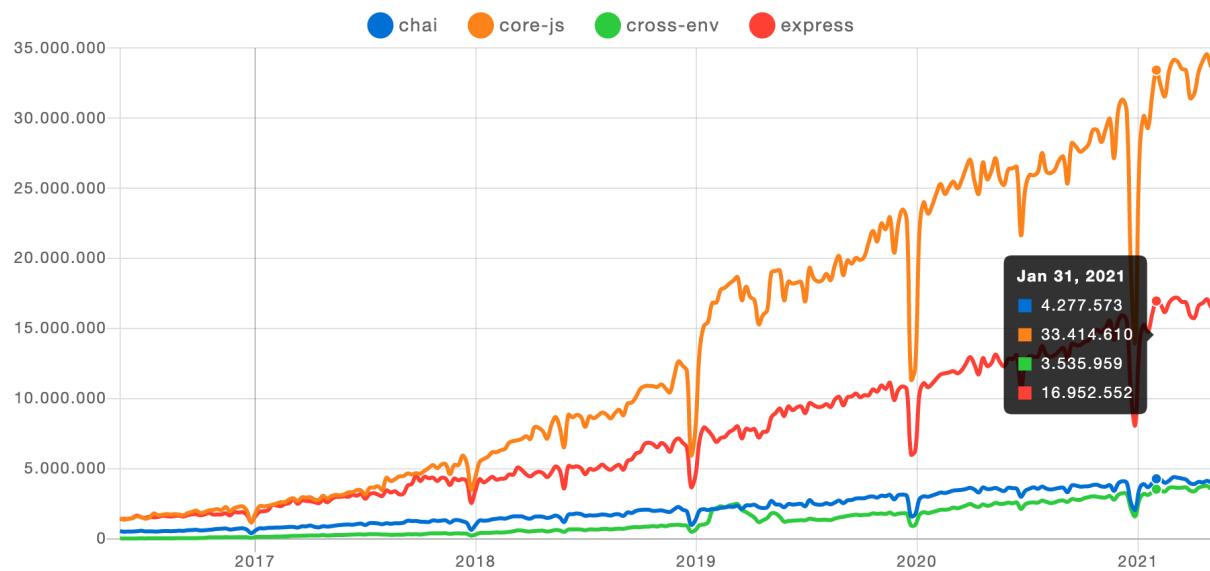


ΕΙΚΟΝΑ 6: ΚΑΜΠΥΛΗ ΛΗΨΗΣ COMPILERS ΔΙΑΣΤΗΜΑ 5 ΕΤΩΝ.

² <https://www.npmtrtrends.com/>

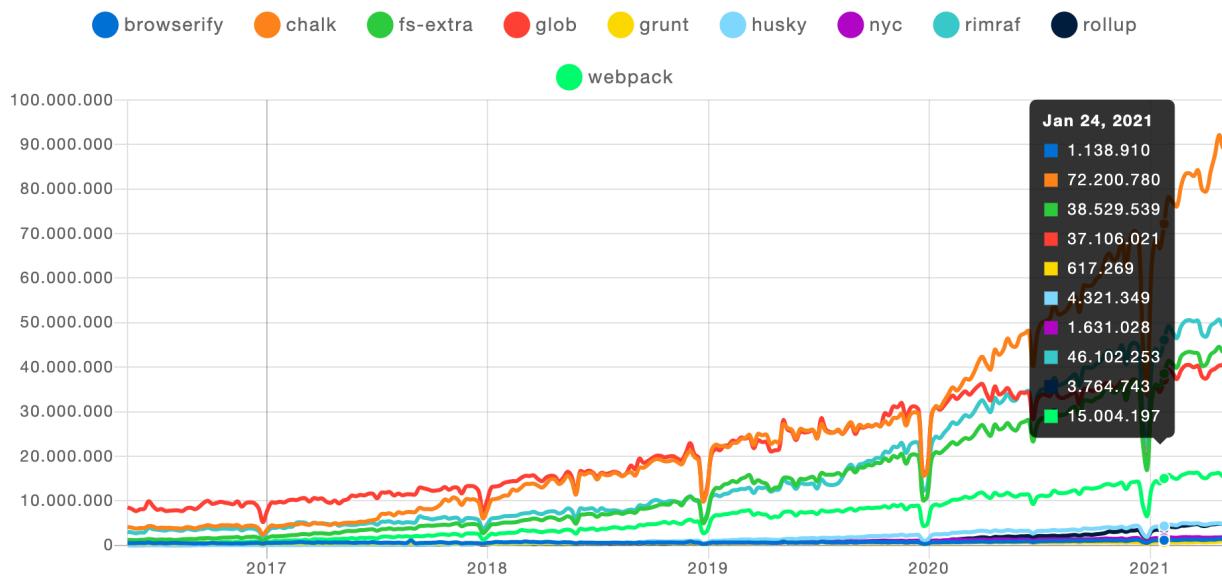


ΕΙΚΟΝΑ 7: ΚΑΜΠΥΛΗ ΛΗΨΗΣ USER INTERFACES ΔΙΑΣΤΗΜΑ 5 ΕΤΩΝ.



ΕΙΚΟΝΑ 8: ΚΑΜΠΥΛΗ ΛΗΨΗΣ FRAMEWORKS ΔΙΑΣΤΗΜΑ 5 ΕΤΩΝ.

Τα δεδομένα αυτά δείχνουν ότι κάποια από τα στοιχεία που εμφανίζονται ως πιο συχνά χρησιμοποιούμενα δεν αποτελούν και αυτά με τον μεγαλύτερο αριθμό λήψεων. Το γεγονός αυτό ενώ αρχικά μας προκάλεσε αμφιβολίες προς την ορθότητα των αποτελεσμάτων μας, μπορεί να ερμηνευτεί με περισσότερους από έναν τρόπους. Τα πακέτα που μελετάμε μπορούν να χρησιμοποιηθούν και αυτούσια χωρίς να είναι μέρος ενός μεγαλύτερου έργου και συνεπώς ο αριθμός των λήψεων δεν εκφράζει το πλήθος και τη συχνότητα επαναχρησιμοποίησης τους. Επιπλέον ο αριθμός των λήψεων ενός λογισμικού δεν αναφέρεται σε μία έκδοση. Έτσι μπορεί κάποιο πακέτο να είναι όντως το πιο συχνά επαναχρησιμοποιούμενο και να ληφθεί μια φορά ενώ κάποιο πακέτο που δεν έχει μεγάλο ποσοστό επαναχρησιμοποίησης να έχει πολλές εκδόσεις και ο προγραμματιστής να χρειάζεται να κάνει διαρκώς λήψη της νεότερης.



ΕΙΚΟΝΑ 9: ΚΑΜΠΥΛΗ ΛΗΨΗΣ INTEROPERABILITY UNITS ΔΙΑΣΤΗΜΑ 5 ΕΤΩΝ.

Ο Πίνακας 5., απεικονίζει τα βασικά στατιστικά στοιχεία που λαμβάνονται διαχωρίζοντας το επαναχρησιμοποιούμενο λογισμικό με βάση τη λειτουργικότητα του. Από τη συγκεκριμένη ανάλυση έχουμε μια ευρεία εικόνα του πλήθους τρίτου λογισμικού που χρησιμοποιεί κατά μέσο όρο κάθε έργο, καθώς και των ακραίων μέγιστων και ελάχιστων τιμών. Επιπλέον παρατηρούμε ότι το μεγαλύτερο πλήθος διαφορετικών στοιχείων επαναχρησιμοποιούμενου λογισμικού ανήκει στην κατηγορία Interoperability Units με την κατηγορία των Compilers να ακολουθεί. Αυτό το στοιχείο δείχνει ότι υπάρχει αφενός μεγαλύτερη ποικιλία σε αυτές τις μονάδες και αφετέρου μεγαλύτερη ευελιξία ως προς την χρήση τους.

Πίνακας 5: Απεικόνιση Πλήθους Χρήσης Τρίτου Λογισμικού ανά τύπο Λειτουργικότητας

Λειτουργικότητα	N	Ελάχιστο Πλήθος	Μέγιστο Πλήθος	Μέσος Όρος
Interoperability Unit	48	17	101	30.04
Compiler	31	19	1157	130.77
Testing unit	24	17	978	111.04
Framework	14	16	87	44.73
User Interface	16	16	171	46.47

Δημιουργείται το ερώτημα πλέον αν οι υπόλοιπες μονάδες λογισμικού δηλαδή τα Frameworks, User Interfaces και Testing Units είναι αυτά που προκαλούν προβλήματα συμβατότητας λόγω της μικρότερης ποικιλίας και τελικά καθορίζουν τον τρόπο εξέλιξης και διαμόρφωσης του λογισμικού.

Με σκοπό να δώσουμε απάντηση στο παραπάνω ερώτημα δημιουργήσαμε τον ακόλουθο πίνακα εξαρτήσεων, χρησιμοποιώντας στοιχεία που παρέχουν οι δημιουργοί των τρίτων λογισμικών στο GitHub. Ο Πίνακας 5., απεικονίζει το πλήθος των εξαρτήσεων των 35 δημοφιλέστερων πακέτων προς άλλα πακέτα, καθώς και το πλήθος των εξαρτώμενων προς αυτά πακέτα.

Από τον πίνακα, Πίνακας 6., το δεδομένο των Dependents επιβεβαιώνει ότι οι Compilers, τα Testing Units και τα Interoperability Units κυριαρχούν στην επιλογή των προγραμματιστών στο σύνολο των έργων λογισμικού JavaScript που φιλοξενεί το GitHub. Συνεπώς το πλήθος των έργων που εξετάζουμε όντως αποτελεί ένα αντιπροσωπευτικό δείγμα και δείχνει πραγματικά τον τρόπο που διαμορφώνεται η ανάπτυξη των εφαρμογών JavaScript. Το δεύτερο δεδομένο του πίνακα που αφορά τις εξαρτήσεις των 35 πακέτων προς τρίτα λογισμικά, χρησιμοποιήθηκε για την εξαγωγή συμπερασμάτων για την μετάβαση στις εκδόσεις λογισμικού. Παρατηρούμε, ότι τα Testing Units και τα User Interface Units συγκεντρώνουν μεγάλο αριθμό εξαρτήσεων, με τα πρώτα να αποτελούν και την πιο δημοφιλή κατηγορία επαναχρησιμοποίησης. Το γεγονός ότι φέρουν τόσες εξαρτήσεις προς τρίτο λογισμικό σε συνδυασμό με την παρατήρηση της μικρότερης ποικιλίας αλλά και τον αριθμό των εξαρτήσεων προς αυτά μας οδηγεί στο συμπέρασμα ότι υπάρχουν πολλοί περιορισμοί που θα πρέπει να ικανοποιούνται για την χρήση τους από κάποιο λογισμικό και άρα τελικά είναι αυτά που καθορίζουν την πορεία εξέλιξης του λογισμικού εφαρμογής.

Πίνακας 6: Απεικόνιση Πλήθους Εξαρτήσεων από και προς Τρίτο Λογισμικό για τα 35 πιο Χρησιμοποιούμενα Στοιχεία

Όνομα Τρίτου Λογισμικού	Dependancies on package.json	Dependents	Λειτουργικότητα
Eslint	83	318.377	Testing Unit
TypeScript	69	294.330	Interoperability Unit
Mocha	106	251.028	Testing Unit
Webpack	90	195.471	Interoperability Unit
Jest	74	177.392	Testing Unit
React	93	173.741	User Interface
Chai	16	160.944	Framework
Babel	49	142.840	Compiler
Prettier	110	142.533	Compiler
Lodash	8	141.513	Compiler
Rimraf	3	133.665	Interoperability Unit
Css-loader	45	101.348	User Interface
Chalk	12	98.067	Interoperability Unit
Rollup	60	90.781	Interoperability Unit
Husky	16	83.543	Interoperability Unit
Gulp	12	79.496	Compiler
Express	48	78.818	Framework
Cross-env	2	76.849	Framework
Sinon	28	69.895	Testing Unit
Fs-extra	13	59.887	Interoperability Unit

Όνομα Τρίτου Λογισμικού	Dependancies on package.json	Dependents	Λειτουργικότητα
Nyc	37	59.771	Interoperability Unit
Vue	74	59.750	User Interface
Lint-staged	31	53.726	Testing Unit
Core-js	56	53.665	Framework
Coveralls	11	52.451	Testing Unit
Karma	65	48.467	Testing Unit
Grunt	22	40.686	Interoperability Unit
Glob	10	35.969	Interoperability Unit
Semver	2	31.446	Compiler
Browserify	61	31.192	Interoperability Unit
Jquery	42	28.998	User Interface
Jasmine	18	26.260	Testing Unit
Uglify-js	2	25.341	User Interface
Postcss	35	17.562	Interoperability Unit
Sass	5		User Interface

Μέρος της πληροφορίας που συλλέξαμε από τα package.json αποτελεί το είδος εξάρτησης που φέρει κάθε έργο προς το τρίτο λογισμικό. Χρησιμοποιώντας αυτό το δεδομένο κατασκευάσαμε τον ακόλουθο πίνακα που μας δίνει μια γενική εικόνα για την χρήση των πακέτων από τα έργα. Ο Πίνακας 7, μας παρέχει πληροφορίες για το πλήθος των εξαρτήσεων στις οποίες συμμετέχει κάθε πακέτο ανά είδος. Συγκεκριμένα παρατηρούμε ότι το μεγαλύτερο πλήθος εξαρτήσεων για τα περισσότερα πακέτα ανήκει στην κατηγορία devDependencies. Όπως αναφέραμε και σε προηγούμενο κεφάλαιο το είδος αυτό αναφέρεται στη χρήση των πακέτων κατά την ανάπτυξη και τον έλεγχο της εφαρμογής. Η παρατήρηση αυτή τονίζει την σημασία και την προσοχή που δίνουν οι προγραμματιστές στον έλεγχο της εφαρμογής και επιβεβαιώνει την βιβλιογραφία που θέλει τους προγραμματιστές να προτιμούν τρίτο λογισμικό κατά τον έλεγχο [7]. Ωστόσο το γεγονός ότι τα ίδια πακέτα έχουν αρκετά μικρό πλήθος dependencies, δηλαδή εξαρτήσεων που αφορούν την παραγωγή του έργου, εγείρει το ερώτημα κατα πόσο τελικά οι προγραμματιστές θεωρούν τα πακέτα αξιόπιστα για την τελική εφαρμογή.

Πίνακας 7: Απεικόνιση Πλήθους Εξαρτήσεων ανά Είδος για τα 35 πιο Χρησιμοποιούμενα Στοιχεία

Όνομα Τρίτου Λογισμικού	Popularity	Συνολικές εμφανίσεις σε όλα τα έργα	dependencies		devDependencies	
			Ανά Σχέση UsedBy	Συνολικά	Ανά Σχέση UsedBy	Συνολικά
Eslint	258	978	31	121	241	857
Babel	216	1157	55	197	203	960
Mocha	194	194	18	18	176	176
Webpack	140	339	24	48	126	291
Chai	117	117	11	11	106	106
TypeScript	108	176	14	23	98	153

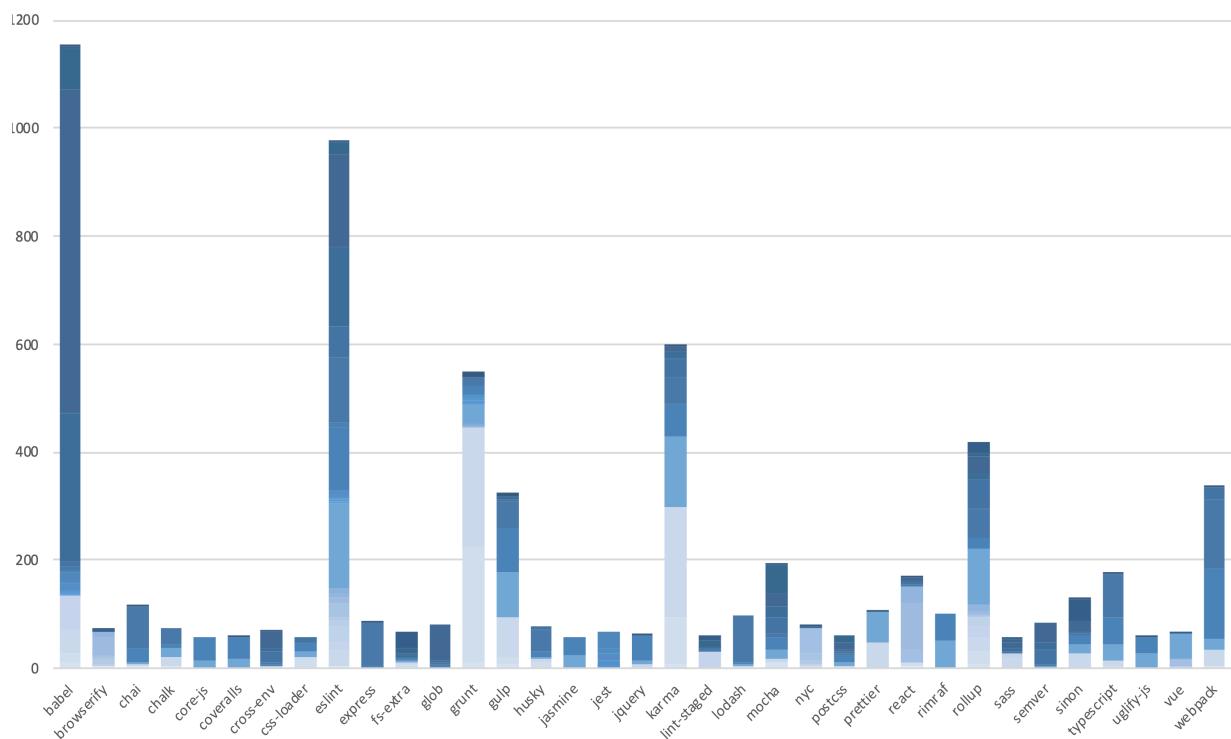
Όνομα Τρίτου Λογισμικού	Popularity	Συνολικές εμφανίσεις σε όλα τα έργα	dependencies		devDependencies	
Karma	106	599	9	49	99	550
Sinon	105	129	11	10	96	119
Prettier	104	105	9	9	96	96
Grunt	102	551	19	86	91	465
Rollup	102	419	7	21	98	398
Rimraf	101	101	20	20	81	81
Lodash	99	99	67	67	32	32
Gulp	86	324	10	30	83	294
Express	86	87	53	53	34	34
Semver	84	84	52	52	32	32
Glob	81	81	31	31	50	50
Husky	79	79	8	8	71	71
Nyc	77	77	3	3	74	74
Chalk	72	73	41	41	32	32
Browserify	72	72	12	12	60	60
React	71	171	49	109	38	62
Cross-env	70	70	7	7	63	63
Jest	68	69	10	10	59	59
Fs-extra	67	68	38	38	30	30
Jquery	63	63	41	41	22	22
Uglify-js	62	62	10	10	52	52
Lint-staged	61	61	4	4	57	57
Coveralls	58	59	3	3	56	56
Core-js	57	57	27	27	30	30
Css-loader	57	57	14	14	43	43
Postcss	49	60	16	20	33	40
Jasmine	45	57	3	4	42	53
Sass	45	57	14	18	33	39
Vue	27	66	12	19	24	47

Σε μεγάλο βαθμό οι προγραμματιστές χρησιμοποιούν τα πακέτα για να επιλύσουν προβλήματα που δημιουργούνται και αφορούν την ανάπτυξη της εφαρμογής και λιγότερο την προώθηση της. Συνεπώς είναι λογικό το μεγαλύτερο πλήθος των εξαρτήσεων να είναι devDependencies μιας και τα περισσότερα προβλήματα υπάρχουν πριν την παραγωγή του τελικού αποτελέσματος. Τέλος όπως ήδη δείξαμε τα πιο δημοφιλή πακέτα ανήκουν στις κατηγορίες Testing Units, Compilers και Interoperability Units και οι τρεις κατηγορίες χρησιμοποιούνται για την ανάπτυξη εφαρμογών, τον έλεγχο του κώδικα και την επίτευξη της συμβατότητας. Οι κατηγορίες αυτές ανήκουν στο κομμάτι της ανάπτυξης του έργου και άρα είναι λογικό οι εξαρτήσεις προς αυτά τα πακέτα να είναι κατά κύριο λόγο devDependencies.

5.1.2 Ερευνητικό ερώτημα 2

[RQ2]: Ποια είναι η επαναχρησιμοποίηση στοιχείων JavaScript ανοιχτού κώδικα σε σχέση με την έκδοσή τους;

Ξεκινώντας την ανάλυση, είναι σημαντικό να αναφερθεί πως κατά την ανάπτυξη εφαρμογών JavaScript συνιστάται η χρήση της πιο πρόσφατης έκδοσης του Node.js ή του υπόλοιπου λογισμικού ώστε να επιλέγουν νεότερες εκδόσεις στα στοιχεία τρίτου λογισμικού ώστε να υποστηρίζουν και υποστηρίζονται από τις νεότερες εκδόσεις Node.js. Η τεχνική διαρκούς αναβάθμισης συστήνεται, επειδή καθιστά τις εφαρμογές πιο ενημερωμένες και σύγχρονες προσφέροντας τη δυνατότητα στον προγραμματιστή να ενσωματώσει ιδιότητες και λειτουργίες που δεν παρέχονται σε προηγούμενες εκδόσεις. Σύμφωνα με τον πρώτο νόμο του Lehman: "ένα σύστημα πρέπει να προσαρμόζεται συνεχώς ειδιάλλως γίνεται προοδευτικά λιγότερο ικανοποιητικό", στο άρθρο "Analyzing the Evolution of Javascript Applications" [5], ο κανόνας αυτός επιβεβαιώνεται μέσω στατιστικής μελέτης και φαίνεται να είναι γνωστός από τους προγραμματιστές που δείχνουν να προτιμούν την προσαρμογή και αναβάθμιση του συστήματος τους έναντι της στασιμότητας. Το κύριο μειονέκτημα αυτής της τεχνικής ωστόσο, αφορά την ενδεχόμενη έλλειψη συμβατότητας μεταξύ των διάφορων στοιχείων που χρησιμοποιούνται στο ίδιο έργο, μετά τη μετάβαση στην τελευταία έκδοση. Επιπρόσθετα, ένα από τα μειονεκτήματα της, είναι η σχέση εξάρτησης που δημιουργείται μεταξύ του προγραμματιστή εφαρμογής και του προγραμματιστή του τρίτου λογισμικού, από την στιγμή που ο πρώτος πρέπει να περιμένει τον δεύτερο για να προβεί σε αλλαγές και αναβάθμιση λογισμικού. Επιλέγοντας διαφορετική προσέγγιση και παραμονή σε παλαιότερες δοκιμασμένες εκδόσεις ο προγραμματιστής πετυχαίνει διατήρηση ενός σταθερού περιβάλλοντος εφαρμογής αλλά θυσιάζει την καινοτομία και την πρόοδο.



ΕΙΚΟΝΑ 10: ΓΡΑΦΗΜΑ ΚΑΤΑΝΟΜΗΣ ΕΚΔΟΣΕΩΝ ΤΡΙΤΟΥ ΛΟΓΙΣΜΙΚΟΥ.

Το Γράφημα, Εικόνα 10., παρουσιάζει την κατανομή των εκδόσεων μεταξύ των διαφορετικών μονάδων τρίτου λογισμικού. Στο διάγραμμα ο χρωματισμός συμβολίζει την έκδοση, η πιο σκούρα απόχρωση αποτελεί την πιο πρόσφατη έκδοση ενώ αντίστοιχα η πιο απαλή την παλαιότερη. Όπως είναι αντιληπτό κάθε στοιχείο ακολουθεί διαφορετική προσέγγιση ως προς την επιλογή των εκδόσεων και δεν παρατηρείται κάποιο καθαρό μοτίβο συμπεριφοράς. Σε κάποια στοιχεία υπάρχουν εκδόσεις με μεγάλη συγκέντρωση, είτε πρόκειται για την πιο πρόσφατη είτε για παλαιότερες, ενώ σε άλλα στοιχεία φαίνεται να υπάρχει μια ισορροπημένη κατανομή στις διαφορετικές εκδόσεις.

Για να γίνει το διάγραμμα πιο κατανοητό και να μπορέσουμε να ερμηνεύσουμε τις πληροφορίες που μας δίνει, ως προς την επιλογή των εκδόσεων, δημιουργήθηκε ο Πίνακας 6., στον οποίο φαίνεται η πιο χρησιμοποιούμενη έκδοση από τα 35 πιο συχνά στοιχεία καθώς και το πλήθος των στοιχείων που τη χρησιμοποιούν.

Πίνακας 8: Περισσότερο Χρησιμοποιούμενη Έκδοση από τα 35 πιο Δημοφιλή Στοιχεία

Όνομα Τρίτου Λογισμικού	Περισσότερο Χρησιμοποιούμενη Έκδοση	Πλήθος	Συνολικός Αριθμός χρήσης	Χαρακτηρισμός έκδοσης
Babel	7.x.x	599	1157	Τελευταία Έκδοση
Eslint	2.x.x	159	978	Προγενέστερη
Karma	1.x.x	205	599	Προγενέστερη
Grunt	1.x.x	222	551	Προγενέστερη
Rollup	2.x.x	105	419	Τελευταία Έκδοση
Webpack	3.x.x	131	339	Προγενέστερη
Gulp	2.x.x	83	324	Προγενέστερη
Mocha	8.x.x	54	194	Τελευταία Έκδοση
TypeScript	4.x.x	80	176	Τελευταία Έκδοση
React	16.x.x	88	171	Προγενέστερη
Sinon	9.x.x	39	129	Προγενέστερη
Chai	4.x.x	77	117	Τελευταία Έκδοση
Prettier	2.x.x	56	105	Τελευταία Έκδοση
Rimraf	3.x.x	51	101	Τελευταία Έκδοση
Lodash	4.x.x	87	99	Τελευταία Έκδοση
Express	4.x.x	81	87	Τελευταία Έκδοση
Semver	7.x.x	38	84	Τελευταία Έκδοση
Glob	7.x.x	66	81	Τελευταία Έκδοση
Husky	4.x.x	40	79	Προγενέστερη
Nyc	15.x.x	46	77	Τελευταία Έκδοση
Chalk	4.x.x	28	73	Τελευταία Έκδοση
Browserify	16.x.x	33	72	Προγενέστερη
Cross-env	7.x.x	33	70	Τελευταία Έκδοση
Jest	26.x.x	33	69	Τελευταία Έκδοση
Fs-extra	9.x.x	30	68	Προγενέστερη
Vue	2.x.x	47	66	Τελευταία Έκδοση

Όνομα Τρίτου Λογισμικού	Περισσότερο Χρησιμοποιούμενη Έκδοση	Πλήθος	Συνολικός Αριθμός χρήσης	Χαρακτηρισμός έκδοσης
Jquery	3.x.x	46	63	Τελευταία Έκδοση
Uglify-js	3.x.x	32	62	Τελευταία Έκδοση
Lint-staged	10.x.x	31	61	Προγενέστερη
Postcss	8.x.x	14	60	Τελευταία Έκδοση
Coveralls	3.x.x	42	59	Τελευταία Έκδοση
Core-js	3.x.x	42	57	Τελευταία Έκδοση
Css-loader	0.x.x	16	57	Προγενέστερη
Jasmine	3.x.x	32	57	Τελευταία Έκδοση
Sass	1.x.x	20	57	Προγενέστερη

Παρατηρούμε ότι υπάρχουν μονάδες τρίτου λογισμικού που το μεγαλύτερο μέρος του συνόλου χρησιμοποιεί μια έκδοση ενώ υπάρχουν άλλες που η πιο επαναχρησιμοποιούμενη έκδοση καθορίζεται από μικρό ποσοστό του συνόλου, γεγονός που επιβεβαιώνει την ύπαρξη διαφορετικών κατανομών όπως παρουσιάζει το Γράφημα. Επιπλέον με τη χρήση του πίνακα γίνεται ξεκάθαρο ότι για τα περισσότερα στοιχεία η τελευταία έκδοση λογισμικού είναι η πιο δημοφιλής.

Συνδυάζοντας την πληροφορία και από τις δύο πηγές εγείρεται το ερώτημα γιατί η κατανομή δεν είναι μοιρασμένη. Γιατί ενώ συνιστάται από τους δημιουργούς η διαρκής μετάβαση σε νέες εκδόσεις και όντως η πιο δημοφιλής είναι κατά κύριο λόγο η τελευταία έκδοση, κάποιοι προγραμματιστές επιμένουν σε παλαιότερες και καθυστερούν την αναβάθμιση. Γιατί δεν συμβαδίζουν όλα τα στοιχεία και ανάλογα με την κατηγορία στην οποία ανήκουν ακολουθούν διαφορετική πορεία.

Με αφορμή τα ερωτήματα που προέκυψαν καταλήξαμε στους παράγοντες που φαίνεται καθορίζουν την επιλογή των εκδόσεων. Ο πρώτος αφορά το επίπεδο δυσκολίας που έχει η μετάβαση σε επόμενες εκδόσεις σε σχέση με τα πρόσθετα οφέλη που προσφέρει στον προγραμματιστή. Σε αρκετές περιπτώσεις οι μεταβάσεις σε επόμενες εκδόσεις τρίτου λογισμικού, απαιτούν από τον προγραμματιστή να επέλθει σε επύπονες και χρονοβόρες αλλαγές στο κύριο λογισμικό, ώστε να είναι συμβατό με τις επιπλέον απαιτήσεις που μπορεί να υπάρχουν, χωρίς όμως να προσφέρονται αντίστοιχα στην εφαρμογή παραπάνω λειτουργίες ή καινοτόμες προσθήκες. Ο δεύτερος παράγοντας και αυτός που φαίνεται να καθορίζει πραγματικά την κατανομή είναι η δυνατότητες συνεργασίας και συνύπαρξης μεταξύ πολλών στοιχείων τρίτου λογισμικού. Η μελέτη μας αφορά εφαρμογές ιστού οι οποίες χρησιμοποιούν διαφορά στοιχεία τρίτου λογισμικού ώστε να δημιουργήσουν το επιθυμητό αποτέλεσμα και να πετύχουν συμβατότητα σε όλες τις πλατφόρμες που εκτελούνται. Συνεπώς, εξήχθη το συμπέρασμα ότι η παραμονή ή μετάβαση σε επόμενες εκδόσεις εξαρτάται από το πλήθος των στοιχείων τρίτου λογισμικού που επαναχρησιμοποιεί η κάθε εφαρμογή. Στα έργα που χρησιμοποιούν μεγάλο πλήθος στοιχείων τρίτου λογισμικού, οι δημιουργοί τείνουν να χρησιμοποιούν παλαιότερες εκδόσεις οι οποίες συνεργάζονται αξιόπιστα και παρέχουν ένα σταθερό ολοκληρωμένο αποτέλεσμα. Στην αντίθετη περίπτωση όταν το πλήθος των στοιχείων τρίτου λογισμικού είναι μικρό, οι προγραμματιστές μεταβαίνουν συνεχώς στην τελευταία έκδοση. Τέλος, η επιλογή αναβάθμισης εξαρτάται και από την σημαντικότητα κάθε στοιχείου στην εφαρμογή. Τα στοιχεία που είναι απαραίτητα και επηρεάζουν τον

κορμό της εφαρμογής ανανεώνονται πιο συχνά από τα στοιχεία που προσθέτουν μικρολειτουργίες ή αφορούν μόνο την εμφάνιση της εφαρμογής και όχι την λειτουργικότητα της.

5.1.3 Ερευνητικό ερώτημα 3

[RQ3]: Ποια είναι η ένταση συνύπαρξης των δημοφιλών επαναχρησιμοποιήσιμων ζευγών στοιχείων ανοιχτού κώδικα JavaScript;

Δεδομένου ότι η JavaScript είναι μια γλώσσα ανάπτυξης ιστού και εκτελείται σε πολλές πλατφόρμες συνδυάζοντας frameworks, plugins και κώδικα τρίτων η συμβατότητα και η σταθερότητα ανεξαρτήτως πλατφόρμας είναι αυτό που προσπαθούν να πετύχουν οι προγραμματιστές. Συνεπώς για να ξεπεραστούν τα προβλήματα συμβατότητας με το περιβάλλον εκτέλεσης και να επιλυθούν ζητήματα συνεργασίας μεταξύ των τρίτων μερών, οι προγραμματιστές είναι ανάγκη να χρησιμοποιούν συνδυαστικά μεταγλωττιστές, ενδιάμεσο λογισμικό και μονάδες δοκιμών. Σε αυτό το ερώτημα θα μελετήσουμε τους πιο δημοφιλείς συνδυασμούς των 35 στοιχείων στα 430 έργα λογισμικού μέσω ανάλυσης σημαντικότητα συσχέτισης και υπολογισμού της έντασης συνύπαρξης.

Αρχικά για τον υπολογισμό της σημαντικότητα συσχέτισης των δημοφιλών ζευγών εκτελέστηκε ανάλυση Spearman, όπως περιγράφηκε στο Κεφάλαιο 3. Στον Πίνακα 9., παρουσιάζεται για κάθε ζεύγος το αποτέλεσμα της Ανάλυσης και η περιγραφή που αποδόθηκε στη σχέση ακολουθώντας την παραπάνω διαδικασία

Πίνακας 9: Απεικόνιση και Περιγραφή Σχέσης Ζευγών από Ανάλυση Spearman

Ζεύγος Στοιχείων	Spearman's rho Correlation Coefficient	Significance (2-tailed)	Περιγραφή σχέσης
babel - webpack	0,619	0,000	
chai - mocha	0,601	0,000	
babel - eslint	0,554	0,000	
chai - sinon	0,478	0,000	
babel - react	0,457	0,000	
eslint - prettier	0,452	0,000	
eslint - webpack	0,408	0,000	
eslint - react	0,400	0,000	
mocha - sinon	0,398	0,000	
react - webpack	0,363	0,000	
babel - rollup	0,348	0,000	
prettier - react	0,305	0,000	
prettier - typescript	0,305	0,000	
eslint - typescript	0,301	0,000	
babel - prettier	0,298	0,000	
babel - typescript	0,280	0,000	
rollup - typescript	0,270	0,000	
chai - karma	0,266	0,000	
react - typescript	0,251	0,000	
karma - rollup	0,221	0,000	
karma - sinon	0,220	0,000	
typescript - webpack	0,220	0,000	
karma - mocha	0,213	0,000	
eslint - rollup	0,202	0,000	
prettier - webpack	0,191	0,000	
prettier - rollup	0,180	0,000	
karma - webpack	0,176	0,000	
grunt - rollup	-0,169	0,000	
babel - grunt	-0,215	0,000	
eslint - grunt	-0,233	0,000	

Ζεύγος Στοιχείων	Spearman's rho Correlation Coefficient	Significance (2-tailed)	Περιγραφή σχέσης
babel - sinon	0,151	0,002	
mocha - typescript	0,148	0,002	
eslint - sinon	0,145	0,003	
grunt - prettier	-0,143	0,003	Αδύναμη αρνητική και σημαντική
grunt - gulp	-0,134	0,005	Αδύναμη αρνητική και σημαντική
eslint - gulp	-0,133	0,006	
sinon - typescript	0,131	0,007	
babel - karma	0,130	0,007	
chai - eslint	0,127	0,008	
grunt - typescript	-0,124	0,010	
grunt - webpack	-0,120	0,013	
chai - typescript	0,117	0,015	
sinon - webpack	0,117	0,016	
karma - typescript	0,104	0,031	
grunt - react	-0,103	0,032	
eslint - mocha	0,101	0,037	
babel - chai	0,097	0,044	
gulp - karma	0,095	0,050	
rollup - webpack	0,095	0,050	
react - sinon	0,092	0,057	
rollup - sinon	0,092	0,058	
gulp - prettier	-0,082	0,091	
react - rollup	0,081	0,092	
chai - webpack	0,074	0,125	
gulp - mocha	-0,072	0,138	
babel - mocha	0,071	0,141	
karma - prettier	-0,069	0,155	
chai - grunt	0,055	0,259	
eslint - karma	0,054	0,267	
chai - gulp	-0,052	0,284	
mocha - webpack	0,046	0,339	
karma - react	-0,046	0,343	
gulp - sinon	-0,045	0,347	
gulp - webpack	-0,041	0,398	
chai - rollup	0,035	0,470	
chai - react	0,029	0,546	
mocha - react	-0,029	0,550	
mocha - prettier	0,029	0,554	
gulp - react	-0,028	0,557	
grunt - karma	0,028	0,558	
mocha - rollup	-0,023	0,631	
grunt - sinon	0,023	0,636	
gulp - typescript	-0,011	0,817	
prettier - sinon	0,010	0,830	
babel - gulp	0,009	0,845	
gulp - rollup	-0,008	0,873	
chai - prettier	0,005	0,914	
grunt - mocha	-0,005	0,917	

Τα αποτελέσματα της ανάλυσης Spearman δείχνουν ότι τα 3 ζεύγη συστατικών με υψηλό επίπεδο συσχέτισης και σημαντικότητας αποτελούνται από:

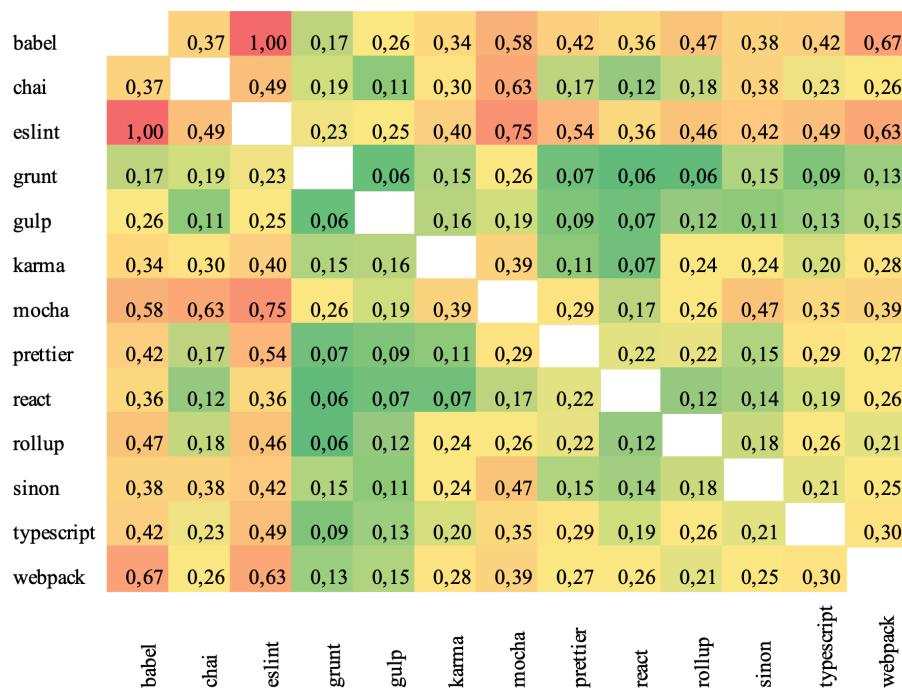
- Compiler και Testing Unit
- Compiler και Interoperability Unit
- Framework και Testing Unit

Μετά τον υπολογισμό της έντασης συνύπαρξης θα περιγράψουμε τα ζευγάρια και την χρήση τους σε JavaScript εφαρμογές.

Στη συνέχεια δημιουργήσαμε έναν πίνακα που απεικονίζει το Popularity of coexistence pairs, την δημοτικότητα ανά ζευγάρι, χρησιμοποιώντας την Σχέση (3) και έπειτα με την Σχέση (4) υπολογίσαμε την ένταση συνύπαρξης, intensity, για όλα τα δημοφιλή ζεύγη και κατασκευάσαμε τον χάρτη της θερμότητας του Σχήματος, Εικόνα 11.

Πίνακας 10: Απεικόνιση Δημοτικότητας ανά ζεύγος στοιχείων τρίτου μέρους

	babel	chai	eslint	grunt	gulp	karma	mocha	prettier	react	rollup	sinon	typescript	webpack
babel	65	175	29	46	60	102	73	63	83	66	73	117	
chai	65		85	33	20	52	110	29	21	31	66	40	46
eslint	175	85		40	44	70	132	94	63	81	74	86	110
grunt	29	33	40		11	27	46	13	10	10	27	16	23
gulp	46	20	44	11		28	34	16	13	21	20	22	27
karma	60	52	70	27	28		68	19	13	42	42	35	49
mocha	102	110	132	46	34	68		50	29	45	83	62	68
prettier	73	29	94	13	16	19	50		38	39	27	50	48
react	63	21	63	10	13	13	29	38		21	24	34	46
rollup	83	31	81	10	21	42	45	39	21		32	46	37
sinon	66	66	74	27	20	42	83	27	24	32		37	44
typescript	73	40	86	16	22	35	62	50	34	46	37		53
webpack	117	46	110	23	27	49	68	48	46	37	44	53	



ΕΙΚΟΝΑ 11: ΧΑΡΤΗΣ ΘΕΡΜΟΤΗΤΑΣ.

Παρατηρούμε ότι τα 4 ζεύγη με την υψηλότερη ένταση συνύπαρξης, intensity, αποτελούνται από:

- Compiler και Testing Unit
- Συνδυασμό Testing Unit
- Compiler και Interoperability Unit

Τα αποτελέσματα των δύο αυτών αναλύσεων δεν μας οδηγούν στο ίδιο συμπέρασμα, καθώς τα 3 κορυφαία ζεύγη διαφοροποιούνται. Καθιστά αυτό όμως λανθασμένη την προσέγγιση μας σε κάποια από τις δύο περιπτώσεις ή το αποτέλεσμα έχει εξήγηση. Η διαφορά οφείλεται στα δεδομένα που χρησιμοποιεί κάθε μία από τις δύο μεθόδους. Στην ανάλυση Spearman η σημαντικότητα της σχέσης υπολογίζεται από τον συνολικό αριθμό των επανεμφανίσεων κάθε στοιχείου ενώ για τον υπολογισμό της έντασης χρησιμοποιούνται οι σχέσεις UsedBy που πραγματοποιεί κάθε στοιχείο. Για να γίνει πιο κατανοητό στον αναγνώστη, για το ζευγάρι eslint-mocha η σχέση χαρακτηρίζεται ως Αδύναμη θετική και χαμηλά σημαντική επειδή το eslint συνολικά εμφανίζεται 978 φορές εκ των οποίων μόνο τις 132 βρίσκεται σε σχέση συνύπαρξης με το mocha και άρα η ανάλυση Spearman δεν κατατάσσει την συσχέτιση σημαντική. Αντίθετα οι σχέσεις UsedBy του eslint είναι 258 εκ των οποίων στις 132 συνυπάρχει με το mocha που συνολικά συμμετέχει σε 194 σχέσεις, συνεπώς η ένταση συνύπαρξης του ζεύγους αγγίζει το 75% και άρα χαρακτηρίζεται υψηλά σημαντική. Αντίστοιχα το ζεύγος mocha-chai που χαρακτηρίζεται από την Spearman ανάλυση με σχέση ισχυρή θετική και υψηλά σημαντική βρίσκεται πιο χαμηλά σε επίπεδο έντασης συνύπαρξης με ποσοστό 67%.

Συνεπώς τα τέσσερα ζευγάρια που απασχολούν την μελέτη είναι:

• **babel-eslint**

Το ζευγάρι αυτό αποτελείται από έναν Compiler και ένα Testing Unit. Αρχικά το babel χρησιμοποιείται κυρίως για τη μετατροπή του κώδικα ECMAScript 2015+ (ES6+) σε μια συμβατή προς τα πίσω έκδοση JavaScript που μπορεί να εκτελεστεί σε όλους τους browsers. Το eslint εντοπίζει συντακτικά λάθη στον κώδικα, λάθη εμφάνισης και τονίζει σημεία που ο κώδικας είναι ανορθόδοξα γραμμένος. Οι λειτουργίες του ζευγαριού γίνονται κυκλικά. Ο linter ελέγχει την ποιότητα του κώδικα πριν την μετατροπή, το babel κάνει την μετατροπή και ξανά το eslint εκτελεί εσωτερικό έλεγχο του κώδικα εφαρμογής.

• **babel-webpack**

Το webpack αναγνωρίζει τα διαφορετικά στοιχεία που χρησιμοποιούνται για την παραγωγή εφαρμογής, διαχωρίζει και ομαδοποιεί τις εξαρτήσεις και παράγει ένα ενιαίο εκτελέσιμο. Στη συνέχεια το babel πραγματοποιεί τις απαραίτητες μετατροπές ώστε το εκτελέσιμο να ανταποκρίνεται στις δυνατότητες και τις απαιτήσεις των περισσότερων browsers.

• **mocha-chai**

Τα δύο αυτά στοιχεία χρησιμοποιούνται μαζί για testing. Το mocha παρέχει σενάρια και εργαλεία ελέγχου, ενώ το chai προσφέρει συναρτήσεις και μεθόδους για σύγκριση αποτελεσμάτων ελέγχου με αναμενόμενα αποτελέσματα. Πρώτα λοιπόν εκτελούνται τα σενάρια του mocha και στη συνέχεια ελέγχονται τα αποτελέσματα από το chai.

• **mocha-eslint**

Και σε αυτό το ζευγάρι σκοπός είναι ο έλεγχος του κώδικα. Η κύρια διαφορά είναι ότι ο linter πρώτα ελέγχει την ποιότητα του κώδικα και στη συνέχεια το mocha εκτελεί σενάρια ελέγχου χρήσης της εφαρμογής.

Στα ζευγάρια συμμετέχουν τα 5 πιο δημοφιλή πακέτα ανάμεσα στα 129 που μελετήθηκαν με το ζευγάρι babel-eslint να έχει και τα 2 πιο επανεμφανιζόμενα στοιχεία στα 430 έργα.

Τα αποτελέσματα επιβεβαιώνουν την βιβλιογραφία που θέλει τους προγραμματιστές να επιλέγουν κατά το testing κυρίως μονάδες τρίτου μέρους καθώς η χρήση τους βελτιώνει την ποιότητα της εφαρμογής και καθιστά το προϊόν πιο αξιόπιστο [7]. Επιπλέον γίνεται κατανοητό ότι οι δημιουργοί προτιμούν τον συνδυασμό μεθόδων ελέγχου ώστε να πετύχουν μεγαλύτερη καλυψιμότητα κώδικα. Η υψηλή ένταση συνύπαρξης μεταξύ των μονάδων ελέγχου και διασυνδεσιμότητας τονίζει την ανάγκη χρήσης λογισμικού για την επίλυση προβλημάτων συμβατότητας μεταξύ άλλων μονάδων τρίτου λογισμικού που χρησιμοποιούνται σε ένα έργο αλλά και μεταξύ των πλατφορμών στις οποίες εκτελείται ο κώδικας εφαρμογής.

Τέλος ο συνδυασμός των ζευγών και η ένταση συνύπαρξης μας δίνει πληροφορίες για τον τρόπο διαμόρφωσης των εξαρτήσεων που δημιουργούνται ανάμεσα στα στοιχεία τρίτου λογισμικού. Οδηγούμαστε στη θεώρηση ότι ο συνδυασμός των ζευγών δεν αποτελεί μόνο επιλογή του προγραμματιστή για την επίλυση προβλημάτων εφαρμογής, αλλά και τα ίδια τα στοιχεία μέσω των εξαρτήσεων τους απαιτούν την δημιουργία των συγκεκριμένων σχέσεων.

Κεφάλαιο 6: Συμπεράσματα και Μελλοντική Επέκταση Έργου

Σε αυτή την ενότητα γίνεται η ανακεφαλαίωση των συμπερασμάτων που προέκυψαν ως αποτέλεσμα της μελέτης καθώς και συζήτηση γύρω από οι πιθανές μελλοντικές επεκτάσεις του έργου.

6.1 Συμπεράσματα

Στη παρούσα εργασία, πραγματοποιήθηκε μελέτη περίπτωσης σε 129 στοιχεία που επαναχρησιμοποιήθηκαν 9389 φορές από 430 εφαρμογές JavaScript που φιλοξενούνται στο αποθετήριο του GitHub. Στόχος μας ήταν ο εντοπισμός των δημοφιλέστερων επαναχρησιμοποιούμενων στοιχείων και η διερεύνηση του τύπου λειτουργικότητας που χρησιμοποιείται κυρίως. Εξετάστηκαν τα επαναχρησιμοποιούμενα στοιχεία σε σχέση με την έκδοση που επαναχρησιμοποίησης. Ως τελικό βήμα, έγινε μελέτη ως προς την ένταση συσχέτισης και επαναχρησιμοποίησης ζευγών. Από την μελέτη έγινε αντιληπτό ότι η συχνότητα επαναχρησιμοποίησης των στοιχεία εξαρτάται από την λειτουργικότητα που προσφέρουν στην παραγόμενη εφαρμογή JavaScript.

Στο RQ1 δείξαμε ότι τα στοιχεία που ανήκουν στις κατηγορίες: Compilers, τα Testing Units και τα Interoperability Units κυριαρχούν στην επιλογή των προγραμματιστών. Επιπλέον στο ίδιο ερώτημα εξετάζοντας το πλήθος των λήψεων πακέτων διαπιστώθηκε ότι δεν αποτελεί δείκτη δημοφιλίας επαναχρησιμοποιούμενων στοιχείων, αλλά δημοτικότητας μεμονωμένων στοιχείων. Το αποτέλεσμα της μετρικής δεν συμπεριλαμβάνει τις εξαρτήσεις λογισμικού και το μεγάλο πλήθος λήψεων ενός πακέτου δεν ξεκαθαρίζει αν πρόκειται για χρήση σε διαφορετικά έργα ή αναβάθμιση λογισμικού.

Στη συνέχεια για το ερώτημα RQ2 εξήχθη το συμπέρασμα ότι η παραμονή ή μετάβαση σε επόμενες εκδόσεις εξαρτάται από το πλήθος των στοιχείων τρίτου λογισμικού που επαναχρησιμοποιεί η κάθε εφαρμογή αλλά και από το πλήθος εξαρτήσεων που έχει κάθε στοιχείο τρίτου λογισμικού. Στα έργα που χρησιμοποιούν μεγάλο πλήθος στοιχείων τρίτου λογισμικού, οι δημιουργοί τείνουν να χρησιμοποιούν παλαιότερες εκδόσεις οι οποίες συνεργάζονται αξιόπιστα και παρέχουν ένα σταθερό ολοκληρωμένο αποτέλεσμα. Στην αντίθετη περίπτωση όταν το πλήθος των στοιχείων τρίτου λογισμικού είναι μικρό, οι προγραμματιστές μεταβαίνουν συνεχώς στην τελευταία έκδοση. Η ίδια αντιμετώπιση υπάρχει και προς το πλήθος εξαρτήσεων. Σε στοιχεία με πολλές εξαρτήσεις οι αλλαγές γίνονται με αργούς ρυθμούς, ενώ για τα στοιχεία που δεν εισάγουν πολλές έμμεσες εξαρτήσεις ισχύει το αντίθετο. Τέλος η επιλογή αναβάθμισης εξαρτάται και από την σημαντικότητα κάθε στοιχείου στην εφαρμογή. Τα στοιχεία που είναι απαραίτητα και επηρεάζουν τον κορμό της εφαρμογής ανανεώνονται πιο συχνά από τα στοιχεία που προσθέτουν μικρολειτουργίες ή αφορούν μόνο την εμφάνιση της εφαρμογής και όχι την λειτουργικότητα της.

Για το ερώτημα RQ3 και την συνύπαρξη των στοιχείων σε ζεύγη παρουσιάσαμε μέσω δύο διαφορετικών αναλύσεων την σημαντικότητα χρήσης μονάδων ελέγχου και μονάδων διασύνδεσης. Δείξαμε ότι τα πιο δημοφιλή ζεύγη συνδυάζουν τις λειτουργικότητες των πιο δημοφιλών μεμονωμένων στοιχείων. Για τα 4 πιο δημοφιλή ζευγάρια περιγράψαμε τα οφέλη χρήσης τους. Τέλος από τα αποτελέσματα τις ανάλυσης, καταλήξαμε στο

συμπέρασμα ότι τα δημοφιλή ζευγάρια δεν δημιουργούνται μόνο χάρη στον καλό συνδυασμό των λειτουργιών που προσφέρουν αλλά επηρεάζονται και από τις εξαρτήσεις των στοιχείων προς άλλα πακέτα.

6.2 Μελλοντικές Επεκτάσεις

Με το πέρας της μελέτης αυτής, νέοι προβληματισμοί δημιουργήθηκαν, που μπορεί να οδηγήσουν σε μελλοντική επέκταση του παρόντος έργου. Σε συνέχεια της παρούσας έρευνας θα θέλαμε να πραγματοποιήσουμε μια πιο εκτενή μελέτη αναφορικά με τις δυνατότητες που προσφέρουν οι πιο δημοφιλείς εκδόσεις λογισμικού τρίτου μέρους και τον τρόπο που η αναβάθμιση σε μεταγενέστερες εκδόσεις επηρεάζει την συνύπαρξη των διαφόρων στοιχείων. Σε επόμενο βήμα θα μπορούσαμε να πραγματοποιήσουμε μια αναζήτησή των μονάδων λογισμικού από τις οποίες εξαρτώνται τα στοιχεία τρίτου μέρους που παρουσιάσαμε. Το αποτέλεσμα αυτής θα οδηγήσει σε έναν αναλυτικό γράφο εξαρτήσεων μέσω του οποίου θα μπορέσουμε να ανακτήσουμε πληροφορίες για τον τρόπο συνύπαρξης των μονάδων λογισμικού, τις ανάγκες διαλειτουργικότητας, το μέγεθος του προβλήματος του dependency-hell και ερωτημάτων που αφορούν την αντιμετώπιση χρήσης έτοιμου κώδικα από τους προγραμματιστές. Επιπλέον ευελπιστούμε σε επόμενη μελέτη σε μεγαλύτερο δείγμα να επαναλάβουμε την διαδικασία εύρεσης έντασης συνύπαρξης και να βρούμε τα 10 πιο διαδεδομένα ζεύγη, την λειτουργικότητα τους και τα οφέλη που προσφέρει η χρήση τους. Τέλος θα θέλαμε να μην περιορίσουμε τον υπολογισμό έντασης συνύπαρξης μόνο σε ζεύγη αλλά να ερευνήσουμε τους πιο δημοφιλείς συνδυασμούς σε πακέτα ανεξαρτήτως πλήθους.

Κεφάλαιο 7: Επίλογος

Το κεφάλαιο αυτό επικεντρώνεται στην παρουσίαση των πιθανών απειλών προς την εγκυρότητα της παρούσας μελέτης, στη σημασία του έργου και στον τρόπο που οι προγραμματιστές ή άλλοι ερευνητές μπορούν να επωφεληθούν από αυτό.

7.1 Απειλές

Σε αυτήν την ενότητα, αναλύονται οι απειλές ως προς την εγκυρότητα της μελέτης που έχουν εντοπιστεί, με βάση την κατηγοριοποίηση που παρουσιάζεται στο [17]. Όσον αφορά την εγκυρότητα δόμησης της έρευνας, πρέπει να αναφερθεί ότι υιοθετήθηκε ένα σύνολο μετρικών επαναχρησιμοποίησης με στόχο την εξαγωγή μετρήσιμων αποτελεσμάτων επαναχρησιμοποίησης στοιχείων εντός ενός οικοσυστήματος φιλοξενίας λογισμικού (δηλαδή Github) [9]. Η λογική πίσω από την επιλογή αυτών των μετρικών βασίστηκε σε ομοιότητες περιεχομένου και πεδίου εφαρμογής με το άρθρο των Kula et al.,[9]. Αποτελεί μελλοντικό στόχο να γίνει αξιολόγηση των μη επιλεγμένων εναλλακτικών μετρικών. Όσον αφορά την εσωτερική εγκυρότητα, σε αυτή τη μελέτη δεν έγινε προσπάθεια προσδιορισμού σχέσεων αιτιότητας, επομένως η απειλή δεν εφαρμόζεται. Όσον αφορά την αξιοπιστία, πιστεύουμε ότι η αναπαραγωγή της έρευνας είναι ασφαλής, καθώς η διαδικασία που ακολουθήθηκε σε αυτήν τη μελέτη έχει τεκμηριωθεί διεξοδικά στα Κεφάλαια 2, 3. Το μόνο μέρος όπου εισάγεται υποκειμενική γνώμη είναι η ταξινόμηση του επαναχρησιμοποιήσιμου στοιχείου σε έναν τύπο λειτουργικότητας. Αυτό έγινε με την απομόνωση λέξεων-κλειδιών από την περιγραφή του στοιχείου στο Github. Όσον αφορά την εξωτερική εγκυρότητα και ιδίως την υπόθεση γενίκευσης, ενδέχεται να προκύψουν αλλαγές στα ευρήματα εάν γίνει αλλαγή στα δείγματα των έργων που μελετήθηκαν. Η μελλοντική αναπαραγωγή αυτής της μελέτης σε άλλα σύνολα έργων JavaScript θα ήταν πολύτιμη για την επαλήθευση αυτών των ευρημάτων.

7.2 Σημασία Έργου

Τα αποτελέσματα αυτής της μελέτης παρέχουν χρήσιμες πληροφορίες και καθοδήγηση σχετικά με τον προγραμματισμό της επαναχρησιμοποίησης στοιχείων στο πλαίσιο της ανάπτυξης εφαρμογών JavaScirpt. Συγκεκριμένα:

Κατά την αναζήτηση ευκαιριών για επαναχρησιμοποίηση τρίτου λογισμικού, μπορεί να εξεταστεί το ενδεχόμενο επαναχρησιμοποίησης στοιχείων που σχετίζονται με διαλειτουργικότητα, μεταγλωττιστές και μονάδες δοκιμών.

- Ανεξάρτητα από τη διαθεσιμότητα, τα στοιχεία που παρουσιάζουν την περισσότερη επαναχρησιμοποίηση είναι κατά μέσο όρο είναι: Compilers, Testing Units και Interoperability Units. Αυτό το γεγονός δείχνει ότι υπάρχει ενδιαφέρον για την επαναχρησιμοποίηση βιβλιοθηκών που μπορούν να ξεπεράσουν το πρόβλημα της συμβατότητας με διαφορετικά προγράμματα περιήγησης, αντί να αντιμετωπίζουν αυτά τα προβλήματα από πρώτο χέρι.
- Αναφορικά με την ανάγκη ενημέρωσης του επαναχρησιμοποιούμενου στοιχείου, διαφοροποιείται μεταξύ έργων που παρουσιάζουν σημαντική επαναχρησιμοποίηση

περισσότερων από 2 στοιχείων και έργων που παρουσιάζουν περιορισμένη επαναχρησιμοποίηση και προτείνεται η υιοθέτηση :

- α) στην πρώτη περίπτωση, της πιο πρόσφατης σταθερής έκδοσης (όχι την τελευταία) για να είναι δυνατός ο αποτελεσματικός χειρισμός των εξαρτήσεων μεταξύ των επαναχρησιμοποιούμενων στοιχείων και
 - β) στη δεύτερη περίπτωση, η μετάβαση στις νέες εκδόσεις και η παρακολούθηση των αλλαγών από τον σχετικό διαχειριστή εξάρτησης (δηλ. ιρπ), στην περίπτωση αυτή ο κίνδυνος είναι περιορισμένος λόγω του μικρού αριθμού των διαφορετικών επαναχρησιμοποιούμενων στοιχείων.
- Όσον αφορά τη δυνατότητα επαναχρησιμοποίησης ζευγών στοιχείων, ο συνδυασμός των Compiler με Testing Units ή Interoperability Units και Frameworks προτείνεται, καθώς όλα λειτουργούν υπό το ίδιο πεδίο, στοχεύοντας στην επίλυση προβλημάτων με στοιχεία τρίτων.
- Με βάση τα αποτελέσματα αυτής της μελέτης περίπτωσης, ενθαρρύνουμε τους ερευνητές να:
- Εκτελέσουν εμπειρικές μελέτες σχετικά με τα επαναχρησιμοποιημένα στοιχεία JavaScript. Επί του παρόντος, υπάρχει ένα μεγάλο αποθετήριο επαναχρησιμοποιήσιμων στοιχείων Ως εκ τούτου, είναι σημαντική η καθοδήγηση της βιομηχανίας λογισμικού σχετικά με τον σχεδιασμό εφαρμογών JavaScript που εκμεταλλεύονται στο έπακρο τα οφέλη της επαναχρησιμοποίησης.
 - Προβούν σε εξέταση της ποιότητας των επαναχρησιμοποιήσιμων στοιχείων για να ελεγχθεί πριν την ενσωμάτωση η πιθανότητα να παρουσιαστούν ευπάθειες που θέτουν σε κίνδυνο τη διαδικασία ανάπτυξης και συντήρησης της υπάρχουσας εφαρμογής.

Παραρτήματα

Σε αυτό το κεφάλαιο έχουμε συμπεριλάβει στιγμιότυπα από τα bash script που δημιουργήθηκαν για τις ανάγκες τις εργασίας και από το τελικό csv που χρησιμοποιήθηκε για την μελέτη.

A. Bash Script

Η δομή που ακολουθεί το πρώτο script είναι:

1. **Λήψη του συμπιεσμένου φακέλου zip που περιέχει το έργο από το GitHub**
2. **Αποσυμπίεση και αποθήκευση καταλόγων σε νέο directory**
3. **Διαχωρισμός και διαγραφή των υποκαταλόγων που δεν περιέχουν αρχεία package.json και των υπολοίπων αρχείων**
4. **Μετονομασία και μετακίνηση όλων των αρχείων package.json σε κεντρικό κατάλογο**
5. **Επεξεργασία των αρχείων package.json. Αφαίρεση ειδικών χαρακτήρων και κενών σειρών**
6. **Δημιουργία csv για κάθε package.json. Το csv περιέχει το όνομα του έργου και την έκδοση του, το όνομα του πακέτου, τα ονόματα των τρίτων λογισμικών με τις εκδόσεις τους και τον τύπο εξάρτησης**
7. **Δημιουργία συνολικού csv για το εκάστοτε έργο από συνένωση των προηγούμενων csv**
8. **Διαγραφή κενών υποκαταλόγων και όλων των αρχείων εκτός του csv**
9. **Τερματισμός καταμέτρησης χρόνου.**

```

1  #!/bin/bash
2  #Μετρητής χρόνου
3  start=$SECONDS
4  #αν δεν υπάρχουν δύο ορίσματα διακόπτεται η εκτέλεση
5  if [ $# != 2 ];then
6  echo "Only works with 2 arguments"
7  exit
8  fi
9
10 cd /Users/MYUSERNAME
11
12 #Λήψη στοιχείων από github
13 wget -r -np -l 1 -A zip https://github.com/$1/$2/tags
14 #Τα στοιχεία αυτόματα αποθηκεύονται σε φάκελο github.com κατά την λήψη
15 cd /Users/MYUSERNAME/github.com/$1/$2/archive
16 #αποσυμπίεση αρχείων και μετακίνηση στον φάκελο εργασίας
17 unzip -q *.zip -d /Users/MYUSERNAME/diplomatiki_files/$2
18 cd /Users/Users/MYUSERNAME/diplomatiki_files/$2
19
20 #διαγράφη των αρχείων που δεν είναι package.json και των κενών καταλόγων
21 find . -not -name "package.json" -type f -delete
22 find . -type d -empty -delete
23
24 #μεταφορά όλων των αρχείων package.json στον κεντρικό κατάλογο μετά από μετονομασία
25 for f in *; do [[ -d "$f" ]] && {
26 dir=$f
27 echo $dir
28 cd $dir
29 s=1
30 find . -name "package*.json" -type f | while read f
31 do
32 newname="package$s.json"
33 mv -n "$f" "$newname"
34 s=$((s+1))
35 done
36 cd ..
37 }; done

```

EIKONA 12: ΣΤΙΓΜΙΟΤΥΠΟ BASH SCRIPT BULLETS 1-4

```

39 #αφαίρεση ειδικών χαρακτήρων από το package.json
40 #η διαδικασία γίνεται με χρήση ενδιάμεσου αρχείου που στο τέλος διαγράφεται
41 find . -name "package*.json" -type f | while read f
42 do
43 touch $f.txt
44 sed -i .bak 's|[",{},]|g' $f >> $f.txt
45 sed -i .bak '/^[[[:space:]]]*$/d' $f >> $f.txt
46 sed -i .bak 's/[[:blank:]]//g' $f >> $f.txt
47 sed -i .bak 's|[---]|g' $f >> $f.txt
48 sed -i .bak 's|[----]|g' $f >> $f.txt
49 sed -i .bak 's|[ ]||g' $f >> $f.txt
50 sed -i .bak 's/\"\\\\\"/g' $f >> $f.txt
51 sed -i .bak 's/[[]]/g' $f >> $f.txt
52 awk '{$1=$1};1' $f >> $f.txt
53 tr '\011' '*' < $f > $f.txt
54 tr -d "----" < $f > $f.txt
55 tr -d "[" < $f > $f.txt
56 tr '\n' '\r' < $f > $f.txt
57
58 while IFS=$\n read -r line;do
59 echo "$line" >> $f
60 done< $f.txt
61 done
62 find . -name "*.bak" -type f -delete
63 find . -name "*.txt" -type f -delete
64 find . -type d -empty -delete

```

EIKONA 13: ΣΤΙΓΜΙΟΤΥΠΟ BASH SCRIPT BULLET 5

```

66 #δημιουργία .csv
67 for f in *; do [[ -d "$f" ]] && {
68   cd $f
69   anasterzia="pack|\"$f\"|.csv"
70   touch $anasterzia
71   #για κάθε package*.json
72   find . -name "package*.json" -type f | while read packe
73   do
74     #·η·: είναι ο διαχωριστικός χαρακτήρας
75     #η πρώτη σειρά του αρχείου περιέχει το όνομα του πακέτου και το θέλουμε
76     a=$(sed -n 1p $packe | cut -d ":" -f2)
77     touch $packe.txt
78     #μας ενδιαφέρουν τα πεδία που αναφέρονται σε εξαρτήσεις
79     #πρώτα διαχωρίζουμε τα dev Dependencies
80     #χρήσιμο ενδιάμεσο temp αρχείου
81     awk '/devDependencies/{flag=1; next;} ./ {flag=$packe | tr ":" "," >> $packe.txt}
82     while IFS=$\r read -r line2 ;do
83       echo '$a,$line2,devDependencies">>$anasterzia
84     done < $packe.txt
85     rm $packe.txt
86   done
87   touch $packe.txt
88   #διαχωρίζουμε τα Dependencies
89   #χρήσιμο ενδιάμεσο temp αρχείου
90   awk '/dependencies/{flag=1; next;} ./ {flag=$packe | tr ":" "," >> $packe.txt}
91   while IFS=$\r read -r line3 ;do
92     echo '$a,$line3,dependencies">>$anasterzia
93   done < $packe.txt
94   rm $packe.txt
95   done
96   done
97   #μετακινούμε όλα τα csv στον αρχικό κατάλογο
98   find . -maxdepth 1 -name "*.csv" -exec mv -{} . \;
99   #διαγράφουμε όλα τα json
100  rm "*.json"
101  cd ..
102  }
103 };done
104
105 #δημιουργόμε 1 συνολικό csv για το έργο
106 touch final_package$1.csv
107 #software->the library or the external soft the project uses
108 echo "project,version,name,software,software_version,type_of_reuse">>final_package$1.csv
109 find . -name "pack*.csv" -type f | while read pack
110 do
111   vers=$(echo $pack| cut -d "|" -f2)
112   while IFS=$\r read -r line1 ;do
113     echo '$1,$vers,$line1">>final_package$1.csv
114   done < $pack
115 done

```

EIKONA 14: ΣΤΙΓΜΙΟΤΥΠΟ BASH SCRIPT BULLETS 6-7

```

117 #διαγράφουμε όλα τα υπόλοιπα csv και τους κενούς καταλόγους
118 find . -not -name "final*" -type f -delete
119 find . -type d -empty -delete
120
121 #σταματάει η χρονομέτρηση
122 if (( $SECONDS > 3600 )); then
123   let "hours=SECONDS/3600"
124   let "minutes=(SECONDS%3600)/60"
125   let "seconds=(SECONDS%3600)%60"
126   echo "Completed in $hours hour(s), $minutes minute(s) and $seconds second(s)"
127 elif (( $SECONDS > 60 )); then
128   let "minutes=(SECONDS%3600)/60"
129   let "seconds=(SECONDS%3600)%60"
130   echo "Completed in $minutes minute(s) and $seconds second(s)"
131 else
132   echo "Completed in $SECONDS seconds"
133 fi

```

EIKONA 15: ΣΤΙΓΜΙΟΤΥΠΟ BASH SCRIPT BULLETS 8-9

Η δομή που ακολουθεί το δεύτερο script είναι:

1. Συνένωση όλων των csv σε ένα τελικό

2. Διαγραφή των μεμονωμένων csv

```
1 #!/bin/bash
2 #Μετρήσεις χρόνου
3 start=$SECONDS
4 cd /Users/MYUSERNAME/diplomatiki_files
5
6
7 touch final_package_all_projects.csv
8 echo "project,version,name,software,software_version,type_of_reuse" >> final_package_all_projects.csv
9 find . -name "*.csv" -exec mv {} /Users/MYUSERNAME/diplomatiki_files\;
10 #αντιγράφουμε το περιεχόμενο κάθε csv στο τελικό
11 find . -name "final_package*.csv" -type f | while read pack
12 do
13 while IFS=$\r read -r line ;do
14 echo "$line" >>final_package_all_projects.csv
15 done < $pack
16 done
17
18 #διαγράφουμε όλα τα υπόλοιπα csv και τους κενούς καταλόγους
19 find . -not -name "final_package_all_projects.csv" -type f -delete
20 find . -type d -empty -delete
21
22 #σταματάει η χρονομέτρηση
23 if (( $SECONDS > 3600 )) ; then
24     let "hours=SECONDS/3600"
25     let "minutes=(SECONDS%3600)/60"
26     let "seconds=(SECONDS%3600)%60"
27     echo "Completed in $hours hour(s), $minutes minute(s) and $seconds second(s)"
28 elif (( $SECONDS > 60 )) ; then
29     let "minutes=(SECONDS/60)/60"
30     let "seconds=(SECONDS%60)%60"
31     echo "Completed in $minutes minute(s) and $seconds second(s)"
32 else
33     echo "Completed in $SECONDS seconds"
34 fi
```

EIKONA 16: ΣΤΙΓΜΙΟΤΥΠΟ BASH SCRIPT BULLETS 1-2

B. Μορφή Csv

Τα δεδομένα που απεικονίζονται στο csv είναι:

- Όνομα Έργου JavaScript ↪ project
- Έκδοση Έργου JavaScript ↪ version
- Όνομα τρίτου λογισμικού ↪ software
- Όνομα-Τίτλος package.json ↪ name
- Έκδοση τρίτου λογισμικού ↪ software_version
- Είδος εξάρτησης προς τρίτο λογισμικό ↪ type_of_reuse

project	version	name	software	software_version	type_of_reuse
zhukov	webogram-0.5.7	Telegram	del	^1.2.0	devDependencies
zhukov	webogram-0.5.7	Telegram	event-stream	^3.1.0	devDependencies
zhukov	webogram-0.5.7	Telegram	gulp	^3.9.0	devDependencies
zhukov	webogram-0.5.7	Telegram	gulp-angular-templatecache	^1.1.0	devDependencies
zhukov	webogram-0.5.7	Telegram	gulp-concat	^2.1.7	devDependencies
zhukov	webogram-0.5.7	Telegram	gulp-gh-pages	^0.5.4	devDependencies
zhukov	webogram-0.5.7	Telegram	gulp-grep-stream	0.0.2	devDependencies
zhukov	webogram-0.5.7	Telegram	gulp-imagemin	^2.3.0	devDependencies
zhukov	webogram-0.5.7	Telegram	gulp-less	^3.0.5	devDependencies

EIKONA 17: ΣΤΙΓΜΙΟΤΥΠΟ ΜΕΜΟΝΩΜΕΝΟΥ CSV

Το τελικό csv το οποίο είναι και αυτό που χρησιμοποιούμε για την εξαγωγή αποτελεσμάτων διαφέρει από τα μεμονωμένα καθώς:

- Το Όνομα τρίτου λογισμικού, software, έχει οριστεί στο πιο επανεμφανιζόμενο όνομα για το εκάστοτε λογισμικό
- Για την Έκδοση τρίτου λογισμικού ,software_version, έχει πραγματοποιηθεί απλοποίηση και συνένωση των υποεκδοσεων στον αριθμό της έκδοσης

project	version	name	software	vers_soft	type_reuse
acornjs	acorn-8.1.0	acorn-benchmarks	acorn	*	dependencies
adobe	brackets-release-1.14.2	brackets-src	acorn	5.x.x	dependencies
angular	material-1.2.2	angular-material-sourc	acorn	7.x.x	dependencies
aui	art-template-4.13.2	art-template	acorn	5.x.x	dependencies
avajs	ava-3.15.0	ava	acorn	8.x.x	dependencies

EIKONA 18: ΣΤΙΓΜΙΟΤΥΠΟ ΤΕΛΙΚΟΥ CSV

C. Αρχεία για επανάληψη μελέτης

Στο προσωπικό repository του συγγραφέα ο ερευνητής που θέλει να επαναλάβει την μελέτη θα βρεί τα Script και τα csv που χρησιμοποιήθηκαν: https://github.com/Anasterzia/Anasterzia_diploma

 Anasterzia Add files via upload	4104219 11 days ago	⌚ 4 commits
lv_csv Add files via upload	11 days ago	
README.md Initial commit	last month	
create_final_csv.sh Add files via upload	last month	
get_edit_script.sh Add files via upload	last month	

EIKONA 19: ΠΕΡΙΕΧΟΜΕΝΟ REPOSITORY

Συγκεκριμένα υπάρχουν τα αρχεία:

- **get_edit_script.sh**

Η χρήση και η περιγραφή του script περιγράφηκε παραπάνω. Ο χρήστης- ερευνητής χρειάζεται να εκτελέσει αυτό το script πρώτο για να ξεκινήσει η δημιουργία των αρχείων csv που χρειάζεται για την μελέτη.

- **create_final_csv.sh**

To script που είναι υπεύθυνο για τη δημιουργία του ενιαίου csv

- **lv_csv**

Ο φάκελος φιλοξενεί όλα αρχεία csv που χρησιμοποιήθηκαν καθώς και τα αρχεία saν που δημιουργεί το spss στα οποία πραγματοποιήθηκαν οι υπολογισμοί και οι στατιστικές μελέτες. Ο ίδιος ο φάκελος αποτελείται από υποφακέλους, που περιέχουν τα ενοποιημένα και χωρισμένα ανά εκατοντάδες έργων csv αρχεία.

	Anasterzia Add files via upload	4104219 11 days ago		History
..				
	diplomatiki_files_100_lastv	Add files via upload	18 days ago	
	diplomatiki_files_200_lastv	Add files via upload	18 days ago	
	diplomatiki_files_300_lastv	Add files via upload	18 days ago	
	diplomatiki_files_400_lastv	Add files via upload	18 days ago	
	diplomatiki_files_500_lastv	Add files via upload	18 days ago	
	500_final.csv	Add files via upload	11 days ago	
	500_final.sav	Add files via upload	11 days ago	

ΕΙΚΟΝΑ 20: ΠΕΡΙΕΧΟΜΕΝΟ ΦΑΚΕΛΟΥ LV_CSV

	Anasterzia Add files via upload	df80b3c 18 days ago		History
..				
	final_composer_all_projects.csv	Add files via upload	18 days ago	
	final_package_all_projects.csv	Add files via upload	18 days ago	

ΕΙΚΟΝΑ 21: ΠΕΡΙΕΧΟΜΕΝΟ ΥΠΟΦΑΚΕΛΩΝ LV_CSV

Βιβλιογραφία

- [1] Abdalkareem, R., Nourry, O., Wehaibi, S., Mujahid, S. and Shihab, E. (2017). "Why do developers use trivial packages? an empirical case study on npm". In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. p.pp. 385–395.
- [2] Alexander, C., Ishikawa, S. and Silverstein, M. (n.d.). A pattern language : towns, buildings, construction. New York: Oxford Univ. Pr.
- [3] Axel Rauschmayer and Amazon.com (Firm (2012). "The past, present, and future of JavaScript : where we've been, where we are, and what lies ahead". Sebastopol, Ca: O'reilly Media.
- [4] Carraz, M., Korakitis, K., Crocker, P., Muir, R. and Voskoglou, C. (2020). Developers Economics: State of the Developer Nation. 18th ed. SlashData Ltd.
- [5] Chatzimpampas, A., Bibi, S., Zozas, I. and Kerren, A. (2019). "Analyzing the Evolution of Javascript Applications". In: Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering.
- [6] Delcev, S. and Draskovic, D. (2018). "Modern JavaScript frameworks: A Survey Study". In: 2018 Zooming Innovation in Consumer Technologies Conference (ZINC). pp.106–109.
- [7] Fard, A.M. and Mesbah, A. (2017). "JavaScript: The (Un)Covered Parts".In: 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST). pp.230–240.
- [8] Gude, S., Hafiz, M. and Wirfs-Brock, A. (2014). "JavaScript: The Used Parts". In: 2014 IEEE 38th Annual Computer Software and Applications Conference. pp.466–475. Available at: .
- [9] Hristov D., Hummel O., Huq M. and Janjic W., "Structuring Software Reusability Metrics for Component-Based Software Development", 7th International Conference on Software Engineering Advances (ICSEA), 2012
- [10] Kikas, R., Gousios, G., Dumas, M. and Pfahl, D. (2017). "Structure and Evolution of Package Dependency Networks". In: Proceedings of the 14th International Conference on Mining Software Repositories (MSR '17).
- [11] Kula, R.G., De Roover, C., German, D.M., Ishio, T. and Inoue, K. (2018). "A generalized model for visualizing library popularity, adoption, and diffusion within a software ecosystem". In: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). pp.288–299.
- [12] Kula, R.G., German, D.M., Ouni, A., Ishio, T. and Inoue, K. (2017). "Do developers update their library dependencies?". Empirical Software Engineering, 23(1), pp.384–417.

- [13] LEACH, R.J. (1996). "Methods of Measuring Software Reuse for the Prediction of Maintenance Effort". *Journal of Software Maintenance: Research and Practice*, 8(5), pp.309–320.
- [14] Li, X., Wang, Z., Wang, Q., Yan, S., Xie, T. and Mei, H. (2016). "Relationship-aware code search for JavaScript frameworks". In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*.
- [15] McIlroy and M.D (1968). "Mass Produced Software Components". *Software Engineering*, NATO Science Committee.
- [16] Mili, H., Mili, F. and Mili, A. (1995). "Reusing software: issues and research directions". *IEEE Transactions on Software Engineering*, 21(6), pp.528–562.
- [17] Paschali, M.-E., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A. and Stamelos, I. (2016). "A Case Study on the Availability of Open-Source Components for Game Development". *Lecture Notes in Computer Science*, pp.149–164.
- [18] Reid, B., Barbosa, K., dAmorim, M., Wagner, M. and Treude, C. (2021). NCQ: code reuse support for Node.js developers.
- [19] Runeson, P. and Höst, M. (2008). "Guidelines for conducting and reporting case study research in software engineering". *Empirical Software Engineering*, 14(2), pp.131–164.
- [20] Zimmermann, M., Staicu, C.-A., Tenny, C. and Pradel, M. (2019). "Small World with High Risks: A Study of Security Threats in the npm Ecosystem". In: *Proceedings of the 28th USENIX Conference on Security Symposium (SEC 19)*. pp.995–1010 .

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

βλπ

βλέπε

κ.λπ.

και λοιπά

Κ.Ο.Κ

και ούτω καθεξής

π.χ.

παραδείγματος χάριν

CSV

comma separated values

DOM

Document Object Model

npm

node package manager

Απόδοση ξενόγλωσσων όρων

Απόδοση

Ασήμαντο
Βιβλιογραφία
Βιβλιογραφικές Βιβλιοθήκες
Βιβλιοθήκες Σεναρίων Ελέγχου
Γεγονός
Διαθεσιμότητα
Δημιουργικό
Δημοφιλία
Δημοφιλής Συνύπαρξη
Δομικό
Δομικό Πλαίσιο
Ένταση Συνύπαρξης
Εξυπηρετητής
Επαναχρησιμοποίηση
Εργαλεία Μορφοποίησης
Κόλαση Εξαρτήσεων
Κόστος
Μεταγλωττιστής
Μεταγλωττιστής Κώδικα σε Κώδικα
Μονάδα Διαλειτουργικότητας
Μονάδα Διεπαφής Χρήστη
Μονάδα Ελέγχου
Ομαδοποιημένες Εξαρτήσεις
Ομαδοποιητής

Ξενόγλωσσος όρος

Trivial
Documentation
Documentation libraries
Test Libraries
Event
Availability
Creational
Popularity
Popularity of coexistence
Structural
Framework
Intensity
Server
Upcycle
Formatting tools
Dependency hell
Price
Compiler
Transpiler
Interoperability Unit
User Interface Unit
Testing Unit
Bundled Dependencies
Bundler

Απόδοση

Ομότιμες Εξαρτήσεις
Ορισμοί τύπων
Πελάτης
Ποιότητα
Πολυπλοκότητα
Προαιρετικές Εξαρτήσεις
Προσαρμογέας
Πυροδοτώ
Σενάριο Φλοιού
Σημαντικότητα
Συμπεριφορικό¹
Σύνθετο
Συντελεστής Συσχέτισης
Χρησιμοποιείται από

Ξενόγλωσσος όρος

Peer Dependencies
Type Definition
Client
Quality
Complexity
Optional Dependencies
Adapter
Trigger
Bash Script
Significance
Behavioural
Composite
Correlation Coefficient
Used By