ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Ανάλυση Ανθεκτικότητας και Επιστημονικότητας Έργων Ανοικτού Κώδικα

ΑΠΟΣΤΟΛΟΣ ΚΡΗΤΙΚΟΣ

ΘΕΣΣΑΛΟΝΙΚΗ

ΣΕΠΤΕΜΒΡΙΟΣ 2023

Το έργο συγχρηματοδοτείται από την Ελλάδα και την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) μέσω του Επιχειρησιακού Προγράμματος «Ανάπτυξη Ανθρώπινου Δυναμικού, Εκπαίδευση και Διά Βίου Μάθηση», στο πλαίσιο της Πράξης «Ενίσχυση του ανθρώπινου ερευνητικού δυναμικού μέσω της υλοποίησης διδακτορικής έρευνας – 2ος Κύκλος» (MIS-5000432), που υλοποιεί το Ίδρυμα Κρατικών Υποτροφιών (ΙΚΥ).

ARISTOTLE UNIVERSITY OF THESSALONIKI

FACULTY OF SCIENCES
SCHOOL OF INFORMATICS

PROGRAM OF POSTGRADUATE STUDIES

PhD THESIS

Open Source Software Resilience and Epistemic Analysis

APOSTOLOS KRITIKOS

THESSALONIKI

SEPTEMBER 2023

This research is co-financed by Greece and the European Union (European Social Fund- ESF) through the Operational Programme «Human Resources Development, Education and Lifelong Learning» in the context of the project “Strengthening Human Resources Research Potential via Doctorate Research – 2nd Cycle” (MIS-5000432), implemented by the State Scholarships Foundation (IKY).

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Ανάλυση Ανθεκτικότητας και Επιστημονικότητας Έργων Ανοικτού Κώδικα

ΑΠΟΣΤΟΛΟΣ ΚΡΗΤΙΚΟΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:

Ιωάννης Σταμέλος, Καθηγητής, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

ΤΡΙΜΕΛΗΣ ΕΠΙΤΡΟΠΗ ΠΑΡΑΚΟΛΟΥΘΗΣΗΣ:

Ιωάννης Σταμέλος, Καθηγητής, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Ελευθέριος Αγγελής, Καθηγητής, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Παναγιώτης Κατσαρός, Αναπληρωτής Καθηγητής, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

ΕΠΤΑΜΕΛΗΣ ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

Ιωάννης Σταμέλος, Καθηγητής, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Ελευθέριος Αγγελής, Καθηγητής, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Παναγιώτης Κατσαρός, Αναπληρωτής Καθηγητής, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Θρασύβουλος - Κων/νος Τσιάτσος, Αναπληρωτής Καθηγητής, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Αλέξανδρος Χατζηγεωργίου, Καθηγητής, Πανεπιστήμιο Μακεδονίας

Σταματία Μπίμπη, Αναπληρώτρια Καθηγήτρια, Πανεπιστήμιο Δυτικής Μακεδονίας

Απόστολος Αμπατζόγλου, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Μακεδονίας

Ημερομηνία Εξέτασης: 29 Σεπτεμβρίου 2023

PhD THESIS

Open Source Software Resilience and Epistemic Analysis

APOSTOLOS KRITIKOS

SUPERVISOR:

Ioannis Stamelos, Professor, Aristotle University of Thessaloniki

THREE-MEMBER ADVISORY COMMITTEE:

Ioannis Stamelos, Professor, Aristotle University of Thessaloniki

Eleutherios Angelis, Professor, Aristotle University of Thessaloniki

Panagiotis Katsaros, Associate Professor, Aristotle University of Thessaloniki

SEVEN-MEMBER EXAMINATION COMMITTEE

Ioannis Stamelos, Professor, Aristotle University of Thessaloniki

Eleutherios Angelis, Professor, Aristotle University of Thessaloniki

Panagiotis Katsaros, Associate Professor, Aristotle University of Thessaloniki

Thrasyvoulos Kon/nos Tsiatsos, Associate Professor, Aristotle University of Thessaloniki

Alexandros Chatzigeorgiou, Professor, University of Macedonia

Stamatia Bibi, Associate Professor, University of Western Macedonia

Apostolos Ampatzoglou, Associate Professor, University of Macedonia

Examination Date: September 29, 2023

ΠΕΡΙΛΗΨΗ

Για περισσότερα από είκοσι χρόνια, το Ελεύθερο Λογισμικό / Λογισμικό Ανοικτού Κώδικα (ΕΛ/ΛΑΚ) έχει εξελιχθεί σημαντικά. Άνοιξε το δρόμο για λύσεις επιχειρήσεων που απέκτησαν παγκόσμια απήχηση και προσέλκυσαν το ενδιαφέρον της Βιομηχανίας. Σε όλο αυτό το διάστημα, έχουν εμφανιστεί πληθώρα μοντέλων αξιολόγησης λογισμικού, τα οποία επικεντρώνονται σε διαφορετικές πτυχές του, όπως η ποιότητα του λογισμικού, η υγεία της κοινότητας, η διακυβέρνηση, οι αδειοδοτήσεις και άλλα. Ορισμένα από αυτά είναι ειδικά διαμορφωμένα για τα έργα ΕΛ/ΛΑΚ. Παρόλα αυτά, ένα καθολικά αποδεκτό μοντέλο αξιολόγησης για το ΕΛ/ΛΑΚ παραμένει ανέφικτο.

Στην παρούσα έρευνα, σκοπεύουμε να προσαρμόσουμε το Πλαίσιο Ανθεκτικότητας των Πόλεων (City Resilience Framework, CRF), ένα πλαίσιο από την περιοχή της Αστικής Ανθεκτικότητας, στα έργα ΕΛ/ΛΑΚ, με στόχο να ενισχύσουμε τη θεωρητική βάση της αξιολόγησης του ΕΛ/ΛΑΚ, επικεντρώνοντας στην ανθεκτικότητα καθώς τα έργα ωριμάζουν. Είναι σημαντικό να σημειώσουμε ότι η πρόθεσή μας δεν είναι να τοποθετήσουμε δίπλα-δίπλα δύο οντότητες ΕΛ/ΛΑΚ ή να τις κατατάξουμε με βάση την ανθεκτικότητα. Αρχικά σχεδιασμένο για να μετρήσει την ανθεκτικότητα των εξελισσόμενων πόλεων, το CRF εμφανίζει εννοιολογικές ομοιότητες με τα έργα ΕΛ/ΛΑΚ. Υποστηρίζουμε ότι ένα μοντέλο αξιολόγησης με επίκεντρο την ανθεκτικότητα μπορεί να συμπληρώσει τα υπάρχοντα μοντέλα που επικεντρώνονται στην ποιότητα και την υγεία του λογισμικού. Η άποψή μας είναι ότι όπως μια πόλη έτσι και ένα έργο ΕΛ/ΛΑΚ παρουσιάζει δυναμική εξέλιξη. Το προτεινόμενο πλαίσιο ενσωματώνει αρμονικά ποσοτικές και ποιοτικές μετρήσεις, επιτρέποντας έτσι την συνεισφορά εμπειρικής γνώσης από εμπειρογνώμονες.

Προκειμένου να μπορέσουμε να πειραματιστούμε με το πλαίσιο μας σε πραγματικά δεδομένα, δηλαδή σε έργα Λογισμικού Ανοικτού Κώδικα, παρουσιάζουμε ένα εργαλείο που χρησιμοποιεί μια σειρά από μετρήσεις που στοχεύουν στην εφαρμογή του προτεινόμενου μοντέλου αξιολόγησης ανθεκτικότητας λογισμικού σε αυτό το διδακτορικό έργο, σε έργα ανοικτού κώδικα λογισμικού. Το εργαλείο λειτουργεί με όλες τις διαστάσεις του έργου, δηλαδή τις δομικές πτυχές, επικεντρώνοντας κυρίως στον πηγαίο κώδικα, τις περιπλοκές των επιχειρηματικών και νομικών σκέψεων, τις πτυχές της ένταξης και τη ζωντανή κοινωνική δυναμική που εκπροσωπείται από την κοινότητα του έργου.

Συνολικά, δοκιμάσαμε αυτό το πλαίσιο στον τομέα του λογισμικού για επιχειρήσεις, αξιολογώντας σημαντικές εκδόσεις έξι έργων ΕΛ/ΛΑΚ, συμπεριλαμβανομένων των Laravel, Composer και PHPMyAdmin. Τα αρχικά μας ευρήματα υπογραμμίζουν τη δυνατότητα του πλαισίου να διακρίνει έργα ανθεκτικά έργα από άλλα που είναι μη ανθεκτικά. Ένα από τα χαρακτηριστικά που διακρίνουν το εργαλείο μας είναι η δυνατότητά του να διασυνδέεται με το απωθετήριο πηγαίου κώδικα, Github. Αυτή η ενσωμάτωση επιτρέπει στο εργαλείο να εξάγει αυτόματα μια πληθώρα μετρήσεων, διασφαλίζοντας ότι η διαδικασία αξιολόγησης είναι ταυτόχρονα αποτελεσματική και εξαντλητική. Ωστόσο, αναγνωρίζοντας τις περίπλοκες πτυχές που συχνά υπονοούνται στα έργα ΕΛ/ΛΑΚ, υπάρχει προβλεπόμενη δυνατότητα

για χειροκίνητη εισαγωγή. Οι ειδικοί, με τη βαθιά τους κατανόηση των λεπτομερειών του έργου, μπορούν να προσφέρουν πρόσθετες πληροφορίες και δεδομένα, διασφαλίζοντας μια ολοκληρωμένη ανάλυση. Μέσω αυτής της αυτοματοποιημένης απόκτησης δεδομένων και εισαγωγής από ειδικούς, το εργαλείο μας στοχεύει σε μια βαθύτερη εξερεύνηση της ανθεκτικότητας ενός έργου ΕΛ/ΛΑΚ.

Εξετάζουμε περαιτέρω μια ακόμη εφαρμογή που θεωρούμε κατάλληλη για την Ανθεκτικότητα του Λογισμικού Ανοικτού Κώδικα. Πρόκειται για μια προσπάθεια να μεταφράσουμε την ανθεκτικότητα που προστίθεται σε ένα ΕΛ/ΛΑΚ μετά από μια συνεισφορά κώδικα, σε μια εικονική ανταμοιβή, που πιστώνεται μέσω blockchain στον χρήστη που συνεισέφερε τον κώδικα.

Η ταχεία εξάπλωση των έργων ΕΛ/ΛΑΚ έχει οδηγήσει σε πληθώρα δεδομένων προσβάσιμων σε μηχανικούς λογισμικού και άτομα που ενδιαφέρονται να μελετήσουν ή να συμμετάσχουν σε τέτοια έργα. Αυτή η διαπίστωση υπογραμμίζει την αυξανόμενη ανάγκη για κριτική ανάλυση και επιβεβαίωση της εγγενούς αξίας ενός έργου ΕΛ/ΛΑΚ μέσα από την επιστημολογική σκοπιά.

Ιστορικά, τα έργα ΕΛ/ΛΑΚ εξυπηρετούν ως πλατφόρμες για την εκμάθηση προγραμματισμού και την καλλιέργεια δεξιοτήτων από μηχανικούς λογισμικού. Πιστεύουμε ότι, μελετώντας σε βάθος τους διαλόγους που συμβαίνουν μεταξύ των μηχανικών λογισμικού, μπορούμε να ανακαλύψουμε νέα γνώση για την καλύτερη αξιολόγηση των προσπαθειών ΕΛ/ΛΑΚ.

Στην στην προσπάθειά μας αυτή, χρησιμοποιούμε την έννοια των επιστημονικών πλαισίων για να εκτελέσουμε Επιστημολογική Ανάλυση Δικτύων (ΕΝΑ). Έχουμε κωδικοποιήσει συζητήσεις από δύο κορυφαίες πλατφόρμες ΕΛ/ΛΑΚ, το LibreOffice και το OpenOffice. Χρησιμοποιώντας το εργαλείο ENA WebKit online, όχι μόνο αναλύσαμε αυτούς τους διαλόγους, αλλά επίσης συγκρίναμε οπτικά τις δομές των δικτύων διαφορετικών μονάδων δεδομένων. Οι εμπειρικές μας έρευνες κάλυψαν τις μέσες δομές δικτύου και των δύο έργων, τα δίκτυα που σχετίζονται με σφάλματα, και διαλόγους επιλεγμένων μηχανικών λογισμικού που συμμετέχουν ενεργά σε συζητήσεις. Τα ευρήματά μας φωτίζουν το επιστημονικό βάθος αυτών των αλληλεπιδράσεων, επισημαίνοντας περαιτέρω την παιδαγωγική τους σημασία.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Λογισμικού

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Λογισμικό Ανοικτού Κώδικα, Ανθεκτικότητα Λογισμικού, Αξιολόγιση Λογισμικού, Εξέλιξη Λογισμικού, Επιστημολογική Ανάλυση Δικτύων

ABSTRACT

For over twenty years, Open Source Software (OSS) has evolved significantly. It paved the way for enterprise solutions that gained traction globally and attracted attention from major industry stakeholders. Over the years, a plethora of software evaluation models have emerged, emphasizing on different aspects like software quality, community health, governance, licensing and so forth. Some of them are specifically tailored for OSS projects. However, a universally endorsed assessment model for OSS remains elusive.

In this research, we aim to adapt the City Resilience Framework (CRF), a framework from the Urban Resilience discipline, for OSS projects, aiming to fortify the theoretical underpinning of OSS assessment, with a focus on resilience as projects mature. It's crucial to note that our intention isn't to juxtapose two OSS entities or rank them based on resilience. Originally conceived to gauge the resilience of evolving cities, the CRF provides an intriguing parallel to OSS projects. We posit that a resilience-centric evaluation model can augment existing models centered on software quality and health. Our perspective is that both cities and OSS projects exhibit dynamic evolution, bearing conceptual similarities. Our advocated framework seamlessly integrates quantitative and qualitative metrics, enhancing its appeal.

In order to be able to experiment with our framework on real data, meaning in Open Source Software projects, we introduce a tool that employs a suite of metrics that aim to apply the proposed software resilience assessment model in this PhD work, to open source software projects. The tool work with all the dimensions of the project, meaning the structural aspects, primarily focusing on the source code, the intricacies of business and legal considerations, the facets of integration, and the vibrant social dynamics represented by the project's community.

In culmination, we've put this framework to the test within the enterprise software realm, assessing significant versions of six OSS projects, including Laravel, Composer, and PHPMyAdmin. Our initial findings underscore the framework's potential to differentiate between projects with varying resilience levels. One of the distinguishing features of our tool is its ability to integrate with Github repositories. This integration allows the tool to automatically extract a wealth of metrics, ensuring that the evaluation process is both efficient and comprehensive. However, recognizing the nuanced complexities that often underlie OSS projects, there's provision for manual input. Experts, with their profound understanding of the project's intricacies, can provide additional insights and data points, ensuring a well-rounded and thorough analysis. Through this symbiosis of automated data acquisition and expert input, our tool aims to a robust and in-depth exploration of OSS project resilience.

We further explore another application we find suitable to Open Source Software Resilience. It is the translation of the resilience added to an OSS after a commit, to virtual tokens, acredited via blockchain to the user that provided the commit.

The swift proliferation of OSS projects has resulted in a surge of data accessible to software engineers and individuals keen on contributing to such projects. This influx underscores the growing necessity to critically analyze and ascertain the inherent value of an OSS project from a knowledge-centric perspective.

Historically, OSS projects have served as platforms for mastering programming and honing software engineering skills. Yet, there's an increasing demand for tangible proof regarding the scientific merits embedded in these projects. We believe that by immersing ourselves in the dialogues shared by software engineers within these community-driven projects and extrapolating insights, we can unearth a novel paradigm for evaluating OSS endeavors.

In our exploration, we are utilizing the concept of epistemic frames to perform Epistemic Network Analysis (ENA). We have coded discussions from two prominent OSS platforms, LibreOffice and OpenOffice. Utilizing the ENA WebKit online tool, we not only dissected these dialogues, but also visually contrasted the network structures of disparate data units. Our empirical investigations spanned across the average network structures of both projects, bug-associated networks, and dialogues of select software engineers actively engaged in discussions. Our findings shed light on the epistemic depth of these interactions, further illuminating their pedagogical significance.

SUBJECT AREA: Software Engineering

KEYWORDS: Open Source Software, Software Resilience, Software Assessment, Software Evolution, Epistemic Network Analysis

ΣΥΝΟΠΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΤΗΣ ΔΙΔΑΚΤΟΡΙΚΗΣ ΔΙΑΤΡΙΒΗΣ

Αυτή η έρευνα εντάσσεται στον τομέα της *Μηχανικής Λογισμικού*, επικεντρώνοντας ιδιαίτερα σε θέματα όπως το *Λογισμικό Ανοικτού Κώδικα (ΕΛ/ΛΑΚ)*, *Ανθεκτικότητα Λογισμικού*, *Εξέλιξη Λογισμικού*, *Αξιολόγηση Λογισμικού* και *Επιστημονική Ανάλυση*.

Ο κύριος μας στόχος είναι η προσαρμογή του Πλαισίου Ανθεκτικότητας των Πόλεων (CRF) με σκοπό τη μέτρηση της ανθεκτικότητας στο πεδίο του Λογισμικού Ανοικτού Κώδικα (ΕΛ/ΛΑΚ). Παράλληλα, εργαζόμαστε στην δημιουργία ενός εργαλείου που θα συγκεντρώνει και θα ενοποιεί διάφορες μετρήσεις που είναι ζωτικής σημασίας για το πλαίσιο ανθεκτικότητας του ΕΛ/ΛΑΚ. Πέρα από την ανάπτυξη του εργαλείου, είναι απαραίτητο για αυτή την έρευνα να εφαρμόσει το νεοσυσταθέν πλαίσιο ανθεκτικότητας σε πραγματικά έργα ΕΛ/ΛΑΚ. Αυτό το βήμα είναι ουσιαστικό για την αξιολόγηση της πρακτικότητας και της αποτελεσματικότητας του πλαισίου σε πραγματικά σενάρια.

Επιπρόσθετα, κατά τη διάρκεια της έρευνας θα εργαστούμε σε ένα πρωτότυπο σύστημα που σκοπεύει να αξιοποιήσει τις λεπτομέρειες του πλαισίου ανθεκτικότητας, με κύριο στόχο την ανταμοιβή των συνεισφερόντων σε έργα ΕΛ/ΛΑΚ. Αυτό που καθιστά αυτό το σύστημα ανταμοιβών ξεχωριστό είναι το γεγονός ότι βασίζεται στην τεχνολογία blockchain.

Τέλος, χρησιμοποιώντας την Επιστημονική Ανάλυση Δικτύων αλλά και τα Επιστημονικά Πλαίσια, στοχεύουμε να εξερευνήσουμε την επιπλέον γνώση που μπορούμε να αντλήσουμε από τον πολυδιάστατο κόσμο της μηχανικής λογισμικού.

Στην συνέχεια παραθέτουμε το χρονοδιάγραμμα των ερευνητικών εργασιών:

1. Ανασκόπηση Βιβλιογραφίας
2. Προσαρμογή του Πλαισίου Ανθεκτικότητας των Πόλεων (CRF) στο Λογισμικό Ανοικτού Κώδικα (ΕΛ/ΛΑΚ)
    - Μελέτη και ανάλυση των συστατικών του Πλαισίου Ανθεκτικότητας των Πόλεων (CRF) και της εφαρμοσιμότητάς του στην ανθεκτικότητα του λογισμικού.
    - Ταυτοποίηση και ορισμός των μετρήσεων από το CRF που μπορούν να αντιστοιχηθούν στην ανθεκτικότητα του λογισμικού στο ΕΛ/ΛΑΚ.
    - Σχεδιασμός του προκαταρκτικού πλαισίου για την ανθεκτικότητα του ΕΛ/ΛΑΚ βασισμένο στο CRF.
3. Δημιουργία Εργαλείου για τη Συνάθροιση Μετρικών
    - Αναγνώριση των κύριων μετρικών απαραίτητων για το πλαίσιο ανθεκτικότητας των OSS.
    - Σχεδίαση της αρχιτεκτονικής του εργαλείου διασφαλίζοντας κλιμακωσιμότητα και προσαρμοστικότητα.

- Ανάπτυξη ενός πρωτοτύπου του εργαλείου για τη συλλογή και τη συγκέντρωση αυτών των μετρικών από διάφορα αποθετήρια OSS.
- Δοκιμή του εργαλείου σε επιλεγμένα έργα OSS για την εξασφάλιση της ορθότητας στην συλλογή μετρικών.

4. Εφαρμογή του Πλαισίου Ανθεκτικότητας σε Έργα OSS

    - Επιλογή έργων OSS για ανάλυση.
    - Εφαρμογή του αναπτυγμένου πλαισίου σε αυτά τα έργα χρησιμοποιώντας το εργαλείο για τη συλλογή των απαραίτητων μετρικών.
    - Ανάλυση των αποτελεσμάτων για την κατανόηση των επιπέδων ανθεκτικότητας αυτών των έργων και τυχόν περιοχών βελτίωσης.

5. Ανταμοιβές Blockchain για Συνεισφέροντες OSS

    - Σχεδίαση ενός μηχανισμού βασισμένου σε blockchain για την ανταμοιβή των συνεισφερόντων με βάση τη βελτίωση της ανθεκτικότητας που φέρνουν τα commits τους.
    - Ενσωμάτωση αυτού του μηχανισμού με το προηγούμενα αναπτυγμένο εργαλείο.
    - Διεξαγωγή πιλοτικών δοκιμών.

6. Επιστημολογική Ανάλυση στην Τεχνολογία Λογισμικού

    - Μελέτη της έννοιας των επιστημονικών πλαισίων εντός του πλαισίου της τεχνικής λογισμικού.
    - Εφαρμογή της Επιστημονικής Δικτυακής Ανάλυσης (ΕΝΑ) σε συζητήσεις και διαλόγους εντός των κοινοτήτων OSS.
    - Ανάλυση των αποτελεσμάτων για την κατανόηση των δομών γνώσης και του επιστημονικού βάθους των αλληλεπιδράσεων.

7. Επιστημονικές εργασίες

    - Συγκέντρωση των ευρημάτων από όλα τα στάδια της έρευνας.
    - Ετοιμασία λεπτομερών αναφορών και ερευνητικών εργασιών για δημοσίευση.
    - Παρουσίαση των ευρημάτων σε σχετικά συνέδρια και εργαστήρια.

Κατά την ολοκλήρωση αυτής της έρευνας, αναμένουμε ένα πλαίσιο, εμπνευσμένο από το Πλαίσιο Ανθεκτικότητας Πόλεων (CRF), προσαρμοσμένο ειδικά για την αξιολόγηση της ανθεκτικότητας στο Λογισμικό Ανοιχτού Κώδικα (ΕΛ/ΛΑΚ). Το θεωρητικό πλαίσιο συμπληρώνεται από ένα ευέλικτο εργαλείο που μπορεί να συγκεντρώνει κρίσιμες μετρήσεις για την αξιολόγηση της ανθεκτικότητας του ΕΛ/ΛΑΚ. Επιπλέον, φιλοδοξούμε να σχεδιάσουμε μια πλατφόρμα που μπορεί να προσφέρει ένα σύστημα ανταμοιβών μέσω blockchain, στους συνεισφέροντες σε έργα ΕΛ/ΛΑΚ. Τέλος, με την επιστημολογική ανάλυση των έργων ΕΛ/ΛΑΚ, προσβλέπουμε

να προσεγγίσουμε νέες δομές γνώσης βασισμένες στις αλληλεπιδράσεις εντός των κοινοτήτων του ΕΛ/ΛΑΚ.

Τα αποτελέσματα αυτής της μελέτης στοχεύουν να εμπλουτίσουν τον ακαδημαϊκό χώρο, αλλά και το ζωντανό οικοσύστημα του ΕΛ/ΛΑΚ, τους συνεισφέροντές του και, κατά συνέπεια, να συμβάλλουν στην παροχή πιο ανθεκτικών εφαρμογών για τη βάση χρηστών του ΕΛ/ΛΑΚ.
Είμαστε περήφανοι που η έρευνά μας συγχρηματοδοτείται από την Ελλάδα και την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο- ΕΚΤ) μέσω του Επιχειρησιακού Προγράμματος «Ανάπτυξη Ανθρώπινου Δυναμικού, Εκπαίδευση και Διά Βίου Μάθηση» στο πλαίσιο του έργου «Ενίσχυση της Ερευνητικής Δυναμικού των Ανθρώπινων Πόρων μέσω Διδακτορικής Έρευνας» (MIS-5000432), που υλοποιείται από το Ίδρυμα Κρατικών Υποτροφιών (ΙΚΥ).

Dedicated to my mother.
Dedicated to my family.

ACKNOWLEDGEMENTS

First and foremost, I wish to express my gratitude to my family. Their unwavering faith and support have been the bedrock of my achievements. The foundation of this work is a testament to their boundless patience and undying love. In moments filled with challenges, as well as those crowned with achievements, their unwavering belief in my capabilities and their consistent encouragement were the guiding lights that illuminated my path, providing clarity even when the journey seemed daunting.

I would like to express my extreme gratitude to Prof. Ioannis Stamelos. His invaluable guidance, mentorship, and depth of wisdom have been instrumental in shaping the contours of this research. Prof. Stamelos possesses a unique knack for asking the right questions, and his keen insights have consistently propelled me towards higher academic excellence. Throughout this journey, his steadfast dedication to academic rigor and his genuine interest in my personal and professional growth have stood as both a source of inspiration, guiding me through the most challenging phases. After 15 years of working together, I can definitely say that who I am, both on a personal and professional level, was hugely impacted by his presence in my life.

I cannot move forward without acknowledging the invaluable feedback and constructive feedback of Prof. Angelis and Prof. Katsaros, esteemed members of my committee. Their expert opinions, combined with their meticulous and thorough reviews, added invaluable depth and perspective to my research, broadening my academic vistas. Their extensive knowledge, coupled with their rich experience, played a pivotal role in refining my thoughts, sharpening my focus, and crystallizing my findings.

In the realm of professional support, I extend my heartfelt thanks to the teams I worked with, during my PhD journey. Social Mind, a company that I build and ran for 8 years, will always hold a special place in my heart. Equally, my current company, Toggl, has been a major enabler for the completion of my PhD. Toggl's continued endorsement and deep understanding of the intricacies and nuances of my academic journey have been of paramount importance, especially during the more intricate, last stage, of my PhD. When one works with a people-first company, fueled with a culture of deep understanding and limiteless support in professional and personal growth, one can only feel creative, safe, powerful.

Furthermore, I wish to convey my heartfelt appreciation to the Faculty of Informatics of Aristotle University of Thessaloniki. Their unwavering support and steadfast belief in my capabilities and vision played an instrumental role in bringing my research to fruition. The resources they generously bestowed upon me went beyond mere physical utilities. They metamorphosed into a sanctuary for my intellectual musings and rigorous academic explorations. I am also deeply appreciative of the financial support they extended whenever possible, allowing me to immerse myself wholeheartedly in my research.

I’d like to also extend my deepest appreciation to each individual who has been a part of this transformative journey. Every single contribution, had a profound impact. Your invaluable contributions, sage advice, and unwavering encouragement have been the cornerstones and guiding stars of this significant chapter in my life.

Lastly I would like to highlight how proud and grateful we are that this research is co-financed by Greece and the European Union (European Social Fund- ESF) through the Operational Programme «Human Resources Development, Education and Lifelong Learning» in the context of the project «Strengthening Human Resources Research Potential via Doctorate Research» (MIS-5000432), implemented by the State Scholarships Foundation (IKY).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Over the past two decades, Open Source Software (OSS) has witnessed profound evolution and growth. Originating as a grassroots movement with the launch of the pioneering Free/Libre Open Source Software operating system, it didn't take long for this nascent trend to captivate developers from every corner of the globe. The momentum was unmistakable. This movement laid the foundation for enterprise-grade solutions, which rapidly gained global traction and magnetized major industry players. This allure is best exemplified by high-profile events such as IBM's acquisition of RedHat. As the years unfolded, a rich tapestry of software evaluation models sprouted, and a significant chunk of these were meticulously tailored to cater to the nuances of OSS projects. Many of these evaluation frameworks primarily focus on the software's quality and its maintainability. In contrast, others are more attuned to assessing the overall health and vitality of OSS initiatives. But, even with such diverse tools at our disposal, a universally accepted and endorsed model for evaluating OSS is yet to see the light of day.

In this research, our endeavor is to adapt and reimagine the City Resilience Framework (CRF), a framework from the realm of Urban Resilience, and sculpt it for OSS projects. Our overarching goal is to bolster the theoretical foundation of OSS assessment, especially emphasizing resilience as these projects advance and mature over time. It's of paramount importance to elucidate that our objective is not to set two OSS entities against each other or to hierarchically rank them solely based on resilience. Our investigative journey is primarily focused on delving deep into the resilience trajectory of an OSS project and meticulously identifying areas of enhancement across four meticulously delineated dimensions: Source Code, Business and Legal, Integration and Reuse, and the pulsating Social (Community) facet. The CRF, originally crafted to assess the resilience of rapidly evolving urban landscapes, offers a compelling and thought-provoking parallel to OSS undertakings. We firmly believe that a resilience-focused evaluation paradigm has the potential to enrich and complement the existing models, which are predominantly anchored in software quality and vitality. It's worth highlighting that while resilience-related concepts, such as sustainability, are sprinkled throughout academic literature, to the best of our knowledge, no currently available model rigorously assesses an OSS project's resilience. Our lens perceives cities and OSS projects as entities in constant flux, exhibiting pronounced resemblances in their evolutionary patterns. The framework we champion seamlessly amalgamates quantitative and qualitative metrics, magnifying its allure and practicality. To validate its efficacy, we've rigorously applied this framework within the expansive domain of enterprise software, meticulously evaluating pivotal versions of six standout OSS projects, including but not limited to, Laravel, Composer, and PHPMyAdmin. Our preliminary insights compellingly indicate the framework's prowess to draw distinctions between projects with diverse resilience gradients.

To practically implement and experiment with our innovative framework using authentic data from Open Source Software projects, we are excited to unveil a tool. This tool is powered by a comprehensive suite of metrics, crafted to infuse the resilience assessment model proposed in this PhD endeavor into open-source software initiatives. This tool is

dexterously designed to work across all facets of a project, encapsulating structural elements, with a pronounced emphasis on the source code, navigating the intricate maze of business and legal paradigms, exploring the multifaceted world of integration, and capturing the effervescent social dynamics emblematic of a project's community.

At the core of our innovative tool lies its distinctive capability to seamlessly integrate with Github repositories. This state-of-the-art integration empowers the tool to autonomously mine and extract an extensive array of metrics. This automation ensures that the evaluation trajectory is not only streamlined but also exhaustive in its scope. Yet, we are acutely aware of the intricate subtleties and layered complexities that frequently characterize OSS projects. To cater to this, our tool boasts a provision that accommodates manual inputs. Seasoned experts, armed with an in-depth comprehension of the project's multifaceted nuances, have the latitude to enrich the analysis by furnishing additional insights and vital data nodes, thereby guaranteeing an analysis that's both holistic and penetrating. It's through this harmonious blend of automated data aggregation and invaluable expert insights that our tool aspires to embark on a vigorous and profound exploration of the resilience intrinsic to OSS projects.

As we delve deeper into our research, we uncover yet another intriguing application that aligns seamlessly with the overarching theme of Open Source Software Resilience. This novel exploration centers around a concept that amalgamates the realms of software development and emerging digital economies. Specifically, we're looking at the mechanism wherein the resilience fortified into an OSS, post a commit, gets a tangible representation. This tangible form manifests itself as virtual tokens. But how are these tokens allocated and authenticated? The answer lies in the revolutionary technology of blockchain.

Upon a successful commit, the resilience bolstered in the OSS is meticulously evaluated and quantified. Following this evaluation, an equivalent amount of virtual tokens is minted. These tokens, bearing intrinsic value derived from the resilience they represent, are then accredited to the respective user responsible for that commit. This entire process, from the evaluation of resilience to the minting and allocation of tokens, is underpinned by the transparent, secure, and immutable nature of blockchain technology.

By integrating blockchain into this process, we not only ensure the authenticity and credibility of the tokens but also provide a digital ledger that chronicles each transaction. This ledger serves as an indelible record, attesting to the contributions made by developers to the OSS resilience. In essence, what we're fostering is a symbiotic ecosystem where software developers are tangibly rewarded for their efforts in enhancing the resilience of Open Source Software, and where each enhancement is securely logged and rewarded in the expansive world of virtual currency.

With the meteoric rise and rapid expansion of OSS projects, there's been an unprecedented influx of data now accessible to software engineers and enthusiastic individuals who harbor aspirations of contributing to these pioneering projects. This deluge of information accentuates the escalating imperative to meticulously scrutinize and accurately gauge the inherent value that an OSS project brings to the table, particularly from a knowledge-driven vantage point.

Historically speaking, OSS projects have always been revered as invaluable platforms that fostered the mastery of programming paradigms and fine-tuning of software engineering prowess. However, in contemporary times, there's a burgeoning demand for tangible evidence that underscores the scientific excellence and rigor embedded within these projects. We are of the firm conviction that by diving deep into the dialogues and discourses exchanged amongst software engineers within these community-centric projects, and by distilling key insights from these conversations, we stand at the cusp of unveiling a groundbreaking framework for assessing OSS initiatives.

The converging domains of learning analytics, data mining, and data science collectively unfurl a rich tapestry of computational and statistical strategies, meticulously crafted to navigate and decode this vast expanse of data. Future sections will delve deeper into the nuances of Epistemic Network Analysis (ENA), an avant-garde network analysis technique that's fast gaining traction amongst a growing cohort of researchers. ENA is envisioned as a potent tool geared towards enabling in-depth analyses anchored in data-rich environments.

Pivotal to the ENA paradigm is the philosophical concept of epistemic frames. This foundational theory elegantly encapsulates the distinctive behaviors, cognitive processes, and strategic actions that are emblematic of a specific community of practitioners. Our investigative approach hones its focus on the epistemic frame that's indigenous to the world of software engineering. We shine a spotlight on its foundational pillars: the vast knowledge repositories, the synthesis of skills and ethical values, and the tactical mechanisms that govern decision-making and rationalization.

In the course of our research journey, we have meticulously coded and analyzed discussions emanating from two prominent OSS platforms, namely, LibreOffice and OpenOffice. By harnessing the capabilities of the ENA WebKit online tool, we embarked on a mission to not only parse these dialogues but also to visually juxtapose the network architectures of varied data subsets. Our empirical endeavors traversed the median network infrastructures of both these projects, networks linked with bug reports, and dialogues curated from select software engineers who were actively involved in vibrant discussions. The insights gleaned from our research cast a spotlight on the profound epistemic layers embedded within these interactions, thereby underscoring their invaluable pedagogical implications.

2. STATE OF THE ART

As we have already mentioned OSS development is considered a mainstream and professional approach in Information Systems. It fuels the development process of several companies and organizations providing source code, testing and bug fixing through its community of developers, translators, testers, and advocates. It is also a driver to the creation of healthy, successful companies [100, 67].

2.1 Urban Resilience and the City Resilience Framework

The City Resilience Framework (CRF), as presented in [41], is the result of research conducted with the aim of establishing an accessible, evidence-based definition of Urban Resilience by the Arup Institute and the Rockefeller Foundation. It takes under consideration the need of cities, as dynamic systems, to be able to adapt and go through challenges while, at the same time, they build resilience in order "to survive in a continuously evolving, uncertain world". It studies the role of city's stakeholders and how their actions may or may not promote the resilience of the city.

The fact that every city is unique, creates a challenge in studying resilience. The authors approach this challenge by defining the City Resilient Index (CRI) which is a set of indicators and variables which allow cities to understand and measure their relative performance regarding resilience. It is worth mentioning and it is also being stressed by the authors of CRF, that the CRI is not aiming to provide a world rank of cities based on their resilience nor a comparing mechanism between the cities. Its main aim is to provide a framework by which a city can better facilitate all those resources, knowledge, and processes to become more resilient over time.

The CRF is, as of the time of writing, actively applied to cities via the 100 Resilient Cities [28], a non profit organization to primarily evaluate the Urban Resilience of more than 90 cities around the world and, additionally, to assist the cities on crises with tailored made resilience strategies.

The CRI suggests four (4) dimensions which are analyzed in twelve (12) goals. The goals are further decomposed to indicators that serve as KPIs while assessing the resilience of a city. The aforementioned structure of the dimensions and goals, which inspired the adaptation to OSS are the following:

1. **Health & well-being:** Related to **people**, working and living in the city. Goals: (1) Minimal human vulnerability, (2) Diverse livelihoods & employment (3) Effective safeguards to human health & life.

2. **Economy & society:** Related to the **organization** of cities on a social and economic level. Goals: (1) Sustainable economy, (2) Comprehensive security & rule of law, (3) Collective identity & community support.

3. **Infrastructure & environment:** Related to **place**, the quality of infrastructure and ecosystems. Goals: (1) Reliable mobility & communications, (2) Effective provision of critical services, Reduced exposure & fragility.

4. **Leadership & strategy:** Related to **knowledge** of the past and adapting appropriately for the future. Goals: (1) Effective leadership & management, (2) Empowered stakeholders, (3) Integrated development planning

In [60] you can find a preliminary work that presented the CRF analyzing the dimensions and goals arguing the conceptual connection of the framework to the OSS domain. In terms of indicators, in [60] only the indicators for the dimensions of Source Code and Business & Legal have been presented. In the next chapter we are going to present, in further detail, the choices we made adapting the CRF and CRI to the proposed Open Source Software Resilience Framework (OSSRF).

2.2 Factors that impact the evolution of an OSS

OSS success was enabled from different factors as literature review shows. Midha and Palvia [74] propose some of these factors such as the type of the license of an OSS project and how permissive it is, community-related aspects such as the number of developers actively working on the project, the number of the end users of the software or the maturity of the localization of the project in several languages. In this work [76], the authors study the aspect of governance of a project, a factor which seems to be important for OSS success. Structural quality remains the first and one of the most extensively studied factors. It has led to the proposal of both generic quality models, like ISO25010 [43], and others, specifically related to OSS, like OpenBRR [97]. Miguel et al. [75] compare the quality models in literature between 1977 and 2013. They categorize them into basic models that evaluate the software product in a holistic way and tailor-made quality models, which extend to component evaluation. The latter category includes the OSS specific models as well. Other studies are extending the research from OSS quality assurance to the social network analysis research field, like in [92], or they are focusing on a thorough study of the project's community in terms of maturity and socio-technical aspects [30]. The authors of [61] have been working with the design of similar models. In their work, we find a wide review of models and approaches for selecting OSS projects that have been published since 2019. Analyzing sixty (60) relevant studies, the authors pinpointed the criteria categories that have been more frequently used, that is, economic data, licensing, community characteristics, application adoption (installability), and support, among others. As Wasserman states in [98], it is important for OSS evaluation models to include, apart from numerical scores and metrics, qualitative criteria as well. Moreover, IT managers need compelling evidence for the resilience of an IT solution before committing themselves and adopting it for their IT architecture. Informal discussions with IT managers revealed that a system is expected to be sufficiently maintained over a period of at least ten (10) years to become eligible for adoption. Finally, this evaluation should be frequently performed as

the OSS project evolves - for example, after each major release -, to be able to observe how its resilience changes over time.

In the following table we summarize literature by referencing research work related to OSS success factors and OSS assessment and evaluation from 1977 to 2023. We would like to highlight that references 7, 10, and 11 are systematic literature reviews published in 2014, 2020, and 2022 respectively.

Reference	Topic	Year of Publication
Wasserman et al[97]	OSS Evaluation	2006
Mobile Vision[76]	OSS Governance	2011
ISO-IEC 25010: 2011[43]	OSS Quality	2011
Midha et al[74]	OSS Success Factors	2012
Miguel et al[75]	OSS Quality	2014
Jansen et al[57]	OSS Health	2014
Texeira et al[92]	Social Network Analysis on OSS ecosystems	2015
Andrade et al[30]	OSS Maturity	2017
Wasserman et al[98]	OSS Evaluation	2017
Lenarduzzi et al[61]	OSS Evaluation	2020
Laila et al[44]	OSS Evaluation	2021
Fang et al[46]	Trust in Software Ecosystem	2022
Laila et al[93]	Mission Critical OSS	2023

Table 2.1: Literature related to OSS success factors and OSS assessment and evaluation between 1977 and 2023

2.3 Open Source Software: The Concepts of Quality, Health, and Resilience

From the literature review summarized in the table of the previous section, we can see that the main focus of the evaluation and assessment models for OSS projects revolves around the concept of software quality. There is also a limited number of works around the concepts of software health and software trust. In [30] the authors highlight how software health is connected to the longevity of an OSS ecosystem and they observe that ”health is typically looked at from a project scope”. In CHAOSS metrics [3], OSS health is associated with social-related aspects.

Axelrode in [31] provides a definition for the resilient software system as follows: A system that “can take a hit to a critical component and recover and come back for more in a known, bounded and generally acceptable period of times”. In Urban Planning and Architecture research field the concept of City Resilience [102] is defined as “the ability [of a system] to cope with change”. City Resilience Framework [41] defines city resilience as “the capacity of cities to function, so that the people living and working in cities – particularly the poor and vulnerable – survive and thrive no matter what stresses or shocks they encounter”.

OSS projects are dynamic systems that are constantly evolving and face changes, be it on a technology level (for example changes in the development stack they are using), on the

governance level (for example changes in the leadership of the project) or on the social level (for example the project's community shifts to another OSS project). Therefore, we find the definition of the CRF, "resilience lies in the ability of a system to suffer stresses and crises and, nevertheless, survive them", to be conceptually relevant to the OSS domain as well.

2.4 Stressors and crises in OSS

Developer or user base loss to a competitive project, unsuccessful major releases, migration or fork of the project from the code development team or parts of it, appearance of new, competitive software applications, hostile behavior by commercial rival solutions, technology evolution that the project fails to follow, or project sustainability issues are only some of the potential crises and stresses an OSS project may face during its life cycle. Here are some examples of OSS projects that have faced crises and stressors.

With Oracle acquiring Sun in 2010, the OpenOffice suite, which was previously acquired by Sun as part of its StarDivision acquisition, became Oracle's property. OpenOffice community saw that as a threat and created a non-profit organization, The Document Foundation. They also forked OpenOffice and created LibreOffice, as a failsafe in case Oracle chose to discontinue OpenOffice as they did with the OpenSolaris operating systems [33]. In this example, we are seeing how a change to an OSS project governance triggers a stressor for the OSS project. Because a for-profit company acquires another company alongside its OSS project OpenOffice, the community of OpenOffice chooses to fork the project and work independently. In addition, the community creates a non-profit foundation to ensure that the newly created fork, LibreOffice, will remain an OSS project. In [48], the authors investigate the case of LibreOffice and how, forking OpenOffice, helped it evolve.

Core-js is an OSS is a well-known universal polyfill of the JavaScript standard library, which provides support for the latest ECMAScript standard and proposals. It is used by companies of significant size like LinkedIn, Netflix, Binance, Spotify, and so forth. The project is being maintained by a community of 112 contributors from which, the founder, is contributing the majority of the commits based on the contributors' insights statistics on the Github repository of the project [7]. Recently the founder of the project published a post [6] in the project's Github repository expressing his concerns that, due to core-js facing sustainability issues, the future of the OSS project is compromised. We consider this example an indicative crisis an OSS project can face. The project lacks a growing community that can ensure the maintainability and evolution of the project. Right now it seems that the founder of the project is the most active developer and that makes him a single point of failure for the longevity of the project.

At this point, we would like to take the chance and clarify why, although the two aforementioned projects are facing stressors and can be considered great candidates for evaluation with our Open Source Software Resilience Framework, are not part of the enterprise testing. In this work, we emphasize presenting the reasoning behind the design of the proposed assessment model and its connection with the CRF. Therefore we chose to test

the framework with a series of OSS projects that we intuitively classify as resilient and non-resilient to validate that our model is successful in distinguishing resilience. We have been documenting the limitations and threats to the validity of our work in the last sections of this manuscript and they will allow us to incrementally build on the foundation of the research provided in this work, in future iterations of our model.

In this scientific work, we are going to base our proposed framework on the City Resilience Framework and present a possible adaptation of it to the OSS domain.

2.5 Different assessment models and metrics aggregations

The open and collaborative nature of Open Source Software (OSS) has been the main driver for software developers, translators, testers, end users, and other members of the community to voluntarily contribute to OSS projects [77]. Lately, we have seen this growth to be accompanied by an increasing adoption of OSS solutions [74] to a variety of sectors, such as industry [77], academia, and the public sector. In 2018, Microsoft acquired Github[1], one of the oldest and most active OSS code repositories, for 7.5 billion dollars [73]. In 2020, the new Open Source Software Strategy 2020 - 2023, published by the European Commission, is entitled “Think Open” [45] and it stresses the necessity to adopt OSS as a means to achieve transparency, open collaboration, interoperability, and cost efficiency in software development.

Every company, organization, government, or academic institution that utilizes OSS projects for their work usually struggle with similar challenges when it comes to the adoption of OSS. What should be the criteria used to select an OSS solution that better suit the organization’s needs when there are several potential candidate OSS projects? Will the project chosen prove to be a resilient or healthy solution? [79] Will it be a sustainable solution that will remain active and keep evolving after 5, 10, or 20 years? The OSS project that we currently use has recently been forked. Should we continue to use the original OSS project or should we migrate to the new, forked, OSS solution? The European Union’s Open Source Software Strategy [45] is indicative of large organizations have concerns similar to the above.

These challenges are relevant to the OSS software assessment process. In [62], the authors have conducted a Systematic Literature Review on Open Source Software Evaluation, Selection, and Adoption with the most recent literature dating 2019 and conclude that OSS software evaluation process is a rather difficult procedure, with a long set of factors to take under consideration.

There are several software evaluation models [75], some of them specifically designed for OSS, in literature. In [60], we proposed an evaluation approach based on the concept of resilience with the aim to study the OSS projects from another point of view. In this work, we present a tool that can be used to investigate these aspects and provide the user with an easy way of assessing OSS project candidates. Currently, Source-o-grapher has been

[1] Github’s Official Website: `https://www.github.com/`

integrated with Github and PHPMetrics library[2] in order to easily assist the assessment of OSS projects implemented in PHP programming language that are published on a Github repository. We selected PHPMetrics as the go-to library because of its open-source nature and also because it was very easy to be integrated with our system. Our system architecture was designed in a way to easily extend to other source code repositories and programming languages, in future extensions.

2.6 Usage & Potential impact of a Resilient based framework to OSS stakeholders

OSSRF was created with several stakeholders in mind. The fact that it focuses on the resilience aspect of an OSS project's evolution, it makes it a good companion with existing OSS assessment models focusing on quality, health, or trust. The use of both quantitative and qualitative indicators also allows experts to be involved in the assessment process which is considered a benefit.

OSS communities and companies that practice OSS, can use OSSRF to frequently monitor their OSS projects for resilience changes. This way they can proactively identify stressors that could hurt the project. For example, if a decrease of resilience on the social level is identified the organization or company can look for recent decisions that might have led members of the team to move away of the OSS project's community. An Inner Source environment could also make use of OSSRF model. Since inner source works similarly with open source, an inner source company could use it to identify resilience changes in its projects. The metrics could be calculated the same way it would happen in a OSS project community (using code analysis tools and the code repository metrics). OSS consultants can be benefited by OSSRF. They can assess the resilience of specific OSS solutions as they evolve, in order to advocate these solutions to their clients. Governments and the public sector can also use OSSRF to validate a proposed OSS solution by a third party in terms of its resilience. The research community, could use resilience as an extra factor when assessing OSS projects. Finally, individual contributors can use OSSRF to assess the resilience of an OSS solution before they join its community or if they want to introduce an OSS tool to the company they are working on the development stack they are using.

2.7 Blockchain: Ethereum & IOST

The term "Blockchain" refers to a record-keeping technology (akin to a database) that is designed with the primary intention of making the system tamper-proof and ensuring its data remains secure and immutable. It is built upon the distributed ledger technology where transactions and their details are recorded simultaneously across multiple points. Each computer participating in a blockchain system retains a copy of these records, preventing potential data loss due to errors, and all copies are updated and verified concurrently. Although this technology resembles a typical database, it varies significantly in its

[2]PHPMetrics Library Official Website: `https://phpmetrics.org/`

data storage and management approach. In a database, information is stored in tables and files, whereas, in a blockchain, it is stored in blocks that are digitally linked, forming a chain—hence the name ”blockchain”. Furthermore, in a blockchain, the management is handled by computers belonging to a peer-to-peer network, contrasting traditional databases managed by a centralized computer.

The aforementioned uses would not have been possible without the innovation introduced by Ethereum. While in the case of Bitcoin, the blockchain is used exclusively for the exchange of cryptocurrencies, essentially serving as a monetary exchange system, Ethereum builds upon blockchain principles and introduces another essential factor to the system: customization through code. This capability is offered through ”smart contracts”. These contracts consist of pieces of code that govern, in the form of an agreement, the actions taking place on a blockchain. Essentially, they ensure that any transaction in the system follows specific rules and do not allow data modification after the process is complete. They themselves constitute a transaction in the system, and naturally, they cannot be modified after their initial definition. This addition, as one can easily understand, has provided new opportunities for leveraging a blockchain. The system no longer only manages cryptocurrencies in the sense of digital money, but also introduces the concept of a ”token”. A token digitizes an existing object or quantitatively expresses theoretical concepts. In essence, it resembles a new currency and can operate based on the principles of a barter economy. There are no restrictions on the type of token as long as there is an appropriate smart contract to define its behavior. Some examples of tokens include concert tickets, legal contracts, and even patients’ medical histories.

IOST provides the capability to implement new smart contracts and immediately publish them to the system. It started as a token on the Ethereum network, but eventually branched off to create a distinct and innovative blockchain platform. Being an open-source project, it allows anyone unrestricted access to its implementation. The source code is readily available on GitHub. One of its advantages is the ease with which someone can implement a blockchain and tailor it to their needs. Moreover, it boasts one of the fastest block creation algorithms, and its smart contracts can be implemented using JavaScript. Furthermore, it has a library, also written in JavaScript, making it the suitable choice in conjunction with the other selected technologies. For the aforementioned reasons, IOST was chosen for the implementation of this specific tool.

2.8 From OSS data to deep learning

A crucial question to answer is how to study deep learning: understanding how an individual adopts the ways of thinking and action that people in a learning culture employ to frame, explore, and solve complex problems. Data science tools enable the analysis of increasingly large volumes of data.

However, if we aim to comprehend and enhance deep learning, we need to employ these advanced technological tools and approaches to understand not only the low-level psychological processes of learning but also how people make sense of what they learn. We

cannot merely ask “What works?” in isolated laboratory settings and under carefully controlled conditions. Instead, we need to understand how learning interventions function—and indeed work—in the real world of school classrooms and beyond, in the worlds of parents, teachers, students, children in creative centers, as well as in online collaboration and learning facilitated by automated pedagogical agents.

This led researchers interested in assessing deep learning to utilize both qualitative and quantitative methods—implying not only methodological pluralism but also active blending in our research. There is a growing literature on mixed methods, but the advent of big data shifts the balance between empirical and theoretical work, urging us to believe that more data can replace deeper understanding.

According to David Williamson Shaffer [88], to assess deep learning, we must see what it means to use large volumes of data in a theoretically robust way. This means something more than simply “mixing” methods or triangulating correlational studies with case studies. Instead, we must understand how to think about empirical approaches in a way that leverages the power of large-scale analyses for Thick Description of Deep Learning.

Data, on their own, lack meaning. Researchers examining data qualitatively will describe their interpretations and ideas as conclusions. Those examining them quantitatively would call them hypotheses. However, in both cases, the point is to understand what is happening. Without an explanation of what we believe is happening, the data themselves have no meaning. When we draw a conclusion or test a hypothesis, we transform data into information: a part of a story about something happening in the world, something that happened, or something we believe is likely to happen.

This is why a situated perspective is so critical. Human actions revolve around symbols: in actions, conversations, written language, as well as in doing things that signify something to themselves and others. The things people say and do are interpreted by others who share their culture. Culture is the way people understand the meaning of things—not just the meaning of the things themselves, but the web of concepts that connect things to each other, things to people, and thus people to each other.

Culture is what turns data into information by adding meaning. If researchers want to understand how deep learning happens and why specific approaches to deep learning work or not, we need a method for analyzing culture on a certain scale to make reasoned analyses of data.

For more than a decade, there has been a movement in social science research towards mixed qualitative and quantitative methods. In such studies, sometimes the work is conducted concurrently, and researchers triangulate findings from one analysis with another. For instance, studies where researchers collect data from surveys and analyze them quantitatively, as well as data from specific target groups, which are analyzed qualitatively. In other studies, researchers use the results of one analysis to inform another (i.e., as information). A handbook describes over 70 different types of mixed methods studies [59].

In chemistry, a mixture is a combination of two elements or compounds in which no substance is changed. If you add meat to a soup, for example, you’re not fundamentally

changing either the meat or the broth. In contrast, a vodka martini is a solution. Once shaken (or stirred, if you prefer), there's no way to separate the vermouth from the vodka.

In approaches that integrate quantitative and qualitative methods, one type of analysis confirms, denies, or extends the other. Ideally, each analysis provides information to the other, in an iterative process. But instead of a realistic mixture of research methods, it's possible to create a research solution where the power of statistics and the power of methods for understanding deep learning are closely interconnected. However, before exploring such an approach, it's important to understand the goals and assumptions of quantitative and qualitative methods.

While there are numerous techniques for quantitative data analysis, statistical tools rely on the idea of sampling. In quantitative analysis, data (the sample) is drawn in an unbiased manner from a larger population, and statistical inquiry answers whether certain characteristics of the sample reflect a characteristic of the larger population. In the context of educational research, the sample is often a collection of individual students, and the population is all students who are "similar to those in the study," which explains why quantitative researchers are concerned about sample selection, self-selection bias, and other factors that could limit the type of students who are "similar" to those in the sample. This also explains why quantitative researchers are concerned with systematic data collection and recording, as statistical tools rely on the presence of stable, well-structured, and ideally comprehensive data.

Quantitative analyses justify claims that observations for a sample are generalized to a population, distinguishing between "real effects" or relationships in the population and random deviations among individuals. In larger samples, the impact of systematic effects, i.e., observations that matter and thus have an effect on the broader population, increases, as systematic effects follow a pattern (hence the term "systematic"), while random effects cancel out. In quantitative analyses, larger samples provide more power as they allow for justifying claims about more effects and relationships in the data.

Such generalization (which is not grounded in the overall context or circumstances) is not the focus of qualitative research, as qualitative research rejects the notion of "real effects" that can be pinpointed in issues or interventions. Instead, qualitative methods assume that observations emerge through possible interactions among participants, contexts, and researchers. This means that a priori determination of the data structure, typically required by quantitative methods, is problematic from an epistemological perspective.

There are various approaches in qualitative research, but its primary goal is to provide some form of what Geertz popularized as "Thick Description": an explanation of how and why events unfold in a specific place and time (Geertz, 1973). Some theorists argue that causal mechanism descriptions are naively realistic, but any qualitative analysis must ensure that it's more of a portrait of participants' experience than a reflection of researchers' preconceptions.

Qualitative researchers use a range of techniques to account for bias—not to eliminate it, which is impossible anyway, but to understand and account for their implications in interpreting data. Grounded Theory data analysis provides a useful mindset for this problem

through the concept of theoretical saturation: an analysis is theoretically saturated when researchers have collected and analyzed sufficient data, such that additional observations confirm existing assumptions rather than lead to new insights [52]. In qualitative analysis, power comes from collecting rich information about a small number of topics, allowing researchers to examine a large set of observations to find consistent interpretations about what people do and why.

Both qualitative and quantitative techniques gain increasing power from collecting more data. However, the concepts of analytical power in qualitative and quantitative research are contradictory. Qualitative analysis seeks to provide a dense description of people and their actions in a specific context, while quantitative analysis aims to provide general claims about subtle differences in observed data. Qualitative analysis requires a large volume of data on individual participants. Quantitative analysis requires data on a large number of individuals. Within the finite resources available in any study, this creates an irreconcilable tension [88].

The data from MOOCs, educational games, simulations, and other computer-based learning environments are often quite rich and capable of enabling descriptions of deep learning for a large number of students. Therefore, they provide an opportunity for learning scientists to combine the tools of sense-making qualitative research to understand deep learning, in relation to quantitative methods, for understanding deep learning at scale. However, to achieve this, researchers studying deep learning must address the technical and epistemological challenge of integrating two distinct analytical frameworks that have remained separate until now.

Since there isn't enough space in a single chapter to present all components of the combination of qualitative and quantitative methods, an attempt will be made to describe the central ideas of a corresponding approach known as quantitative ethnography. Quantitative ethnography starts with the assumption that each learning culture is characterized by a Big-D Discourse (with capital 'D'), defined by Gee as a particular way of "speaking, listening, writing, reading, acting, interacting, believing, valuing, and feeling (as well as using various objects, symbols, images, tools, and technologies)" [50]. A Big-D Discourse is a communication pattern within a community that shapes how a group perceives the meanings of the world. Deep learning requires understanding and learning more than just basic elements and skills. This means understanding how to speak, think, and act (as well as valuing, feeling, and making decisions) like an expert in a field. Deep learning involves learning a Big-D Discourse, which means learning how to make meaning and perceive the world in a specific way.

As researchers aiming to understand and enhance deep learning, we must assess the extent to which students can engage in a Big-D Discourse. However, we can't directly observe the Big-D Discourse. Instead, we have what Gee calls small-d discourse: the things people actually say, taking into account that people express themselves through "body, clothing, non-linguistic symbols, objects, tools, technologies, but also times and places" as well as "ways of acting and interacting" (Gee, 1999). The "small discourse" is the observable manifestation of communication: what we can observe about how people interact.

To understand a learning culture, researchers must find a way to move from small-d discourse to Big-D Discourse—to extract specific things that people said and did and find their meanings. And a key part of how qualitative researchers make sense of a culture is through the process of coding.

A Code (with capital 'C') describes the culturally relevant meaning of an event. Goodwin argues that learning in any domain involves the development of professional vision: "socially organized ways of seeing and understanding events that are responsible for the distinct interests of a particular social group" [53]. In other words, absorbing a culture requires learning its Codes.

Different researchers, working in different environments, using different types of data or employing different qualitative analysis tools, use various techniques to identify Codes within a learning culture. However, the goal of qualitative analysis is almost always to develop Codes that (a) are grounded in things that matter to participants in the field and (b) emerge from the data themselves rather than from existing theories of the researchers [39], [82]. Once a researcher identifies a set of culturally relevant Codes in the data, they must then determine a systematic way of identifying these Codes in the data. Each Big-C Code in the Big-D Discourse requires a small-c-code that describes what it counts as evidence in the small-d-discourse for the Big-C-Code. This is, of course, just a more technical way of saying that qualitative researchers construct codebooks that define their Codes and show how to apply those Codes to their data.

However, while understanding a Discourse requires understanding the Codes, it's not enough to merely identify the Codes in the data. Qualitative researchers create thick descriptions, understanding how the Codes are systematically related to each other. Thus, researchers can understand a culture by analyzing how the Codes within a Discourse are systematically interrelated.

Some researchers use the method of Epistemic Network Analysis (ENA). ENA is a network analysis technique designed to model qualitative data by examining how codes and, by extension, Codes are systematically related to each other in discourse. In the following chapters, we will discuss this technique in more detail.

3. RESEARCH PLAN

This research work is set within the domain of *Software Engineering*, with a key focus on topics including *Open Source Software (OSS)*, *Software Resilience*, *Software Evolution*, *Software Assessment*, and *Epistemic Analysis*.

3.1 Objectives

Our primary objective remains steadfastly centered on the adaptation of the City Resilience Framework (CRF) with a specific aim of gauging and amplifying resilience within the realm of Open Source Software (OSS). This endeavor is not solitary. Parallel to this, we are fervently working on conceptualizing a tool to aggregate and consolidate various metrics that stand pivotal for the comprehensive OSS resilience framework. Beyond the tool's development, it becomes imperative for this research to rigorously apply the newly crafted resilience framework to real-world OSS projects. This step is quintessential to assess the practicality and effectiveness of the framework in dynamic, real-world scenarios.

Moreover, this research will branch into a novel direction by pioneering the creation of a unique mechanism. This mechanism is poised to leverage the intricacies of the resilience framework, with its primary goal being to reward the contributors of OSS. What makes this reward system stand out is its foundation in blockchain technology. The rewards will not merely be based on volume but will keenly consider the magnitude of resilience enhancement that each contribution introduces to the project.

To top off our ambitious research trajectory, we are poised to immerse ourselves in the profound depths of epistemic analysis. By harnessing the nuanced capabilities of both Epistemic Network Analysis and Epistemic Frames, we aim to bring forth insights tailored for the multifaceted world of software engineering, offering a fresh perspective to the domain's complexities.

3.2 Research activities timeline

1. Literature review
2. Adaptation of the City Resilience Framework (CRF) to Open Source Software (OSS)
    - Study and analyze the City Resilience Framework's components and its applicability to software resilience.
    - Identify and define metrics from the CRF that can be mapped to software resilience in OSS.
    - Design the preliminary framework for OSS resilience based on the CRF.
3. Creation of Tool to Aggregate Metrics

- Identify the primary metrics necessary for the OSS resilience framework.
- Design the architecture of the tool ensuring scalability and adaptability.
- Develop a prototype of the tool to collect and aggregate these metrics from various OSS repositories.
- Test the tool on select OSS projects to ensure accurate metric collection.

4. Application of the Resilience Framework to OSS Projects
    - Select a diverse range of OSS projects for analysis.
    - Apply the developed framework on these projects using the tool to collect necessary metrics.
    - Analyze the results to understand the resilience levels of these projects and any potential areas of improvement.
5. Blockchain Rewards for OSS Contributors
    - Design a blockchain-based mechanism to reward contributors based on the resilience improvement their commits bring.
    - Integrate this mechanism with the previously developed tool.
    - Run pilot tests to ensure the transparent and fair distribution of rewards.
6. Epistemic Analysis in Software Engineering
    - Study the concept of epistemic frames within the software engineering context.
    - Apply Epistemic Network Analysis on discussions and dialogues within OSS communities.
    - Analyze the results to understand the knowledge structures and the epistemic depth of interactions.
7. Consolidation and Reporting
    - Consolidate findings from all phases of the research.
    - Prepare detailed reports and research papers for publication.
    - Present findings in relevant conferences and workshops.

3.3 Expected Outcomes

At the culmination of this research, we anticipate a variety of outcomes. Foremost, we expect a framework, inspired by the City Resilience Framework, tailor-made for assessing resilience in Open Source Software. Complementing this, a versatile tool capable of aggregating pivotal metrics for OSS resilience assessment will be introduced. Additionally, the blockchain module aims to be a platform that can offer transparent, equitable,

and merit-based reward system for OSS contributors. Last but not least, the epistemic analysis is set to provide profound insights into the very knowledge structures and depths of interactions within the OSS communities.

This research is poised to bring groundbreaking insights into software resilience within the expansive realm of Open Source Software. The outcomes of this study are aiming to enrich the academic sphere but also the vibrant OSS ecosystem, its contributors and, therefore, contribute in providing more resilient applications for the OSS user base. With Epistemic Analysis we are aiming in deep diving in the vast pool of Open Source data getting interesting insights for the OSS projects themselves but, at the same time, creating a knowledge base and a process that can assist the education process in software engineering.

We are proud that our research is co-financed by Greece and the European Union (European Social Fund- ESF) through the Operational Programme «Human Resources Development, Education and Lifelong Learning» in the context of the project «Strengthening Human Resources Research Potential via Doctorate Research» (MIS-5000432), implemented by the State Scholarships Foundation (IKY).

4. OPEN SOURCE SOFTWARE RESILIENT FRAMEWORK

We were inspired by the CRF because we believe that OSS projects share conceptual similarities with cities. They are, as well, dynamic and continuously evolving systems. They have their own structural properties, which affect their robustness and ultimately their ability to last, hence they affect their resilience. They attract people around them who form communities. When those communities flourish, they usually need, as it happens with cities, a governance model. Both cities and OSS projects might face stresses and crises. Sometimes, these challenges might endanger their very survival, depending on the severity and duration of the challenge.

In this section we are presenting our attempt to adapt the CRF to OSS projects. We aim in utilizing the models and metrics proposed in the extensive literature regarding software quality, metrics and evaluation, to propose a framework that will assess the relative performance of an OSS project towards resilience. At this point we would like to note that this scientific work provides one way of adapting the CRF to OSS and that it was designed under the subjective lens of the authors. Other interpretations and adaptations of CRF are possible. However, each one must be validated with actual resilient or non-resilient OSS projects.

OSSRF follows the architecture of CRF. Its structure consists of three layers. At the first layer we have the four (4) key dimensions: Source Code, Business & Legal, Integration & Reuse and Social (Community). Those dimensions are further analyzed to twelve (12) goals. Finally, goals are further decomposed to indicators to provide a way of measuring the performance of the OSS project in each of the goals and dimensions, going bottom up. You can find a visual representation of the OSSRF showing the first two levels of the framework in Figure 4.1.

4.1 The Source Code Dimension (D01)

We argue that for an OSS project, the source code (for example files, classes, and so forth) is the structural unit of the project. Given the vast application of OSS, source code related aspects, like its architecture, ability to be maintained or how secure or tested it is, can be affected by a series of factors like the source code language or the software development style. In [29] the authors study the design quality of OSS projects in several domains and find that the quality of design of OSS projects varies, indeed, between software domains. In Figure 4.2 we can see a detailed analysis of the dimensions including the relevant goals and indicators.

4.1.1 Goals

In OSSRF we propose the following goals for the Source Code Dimension:

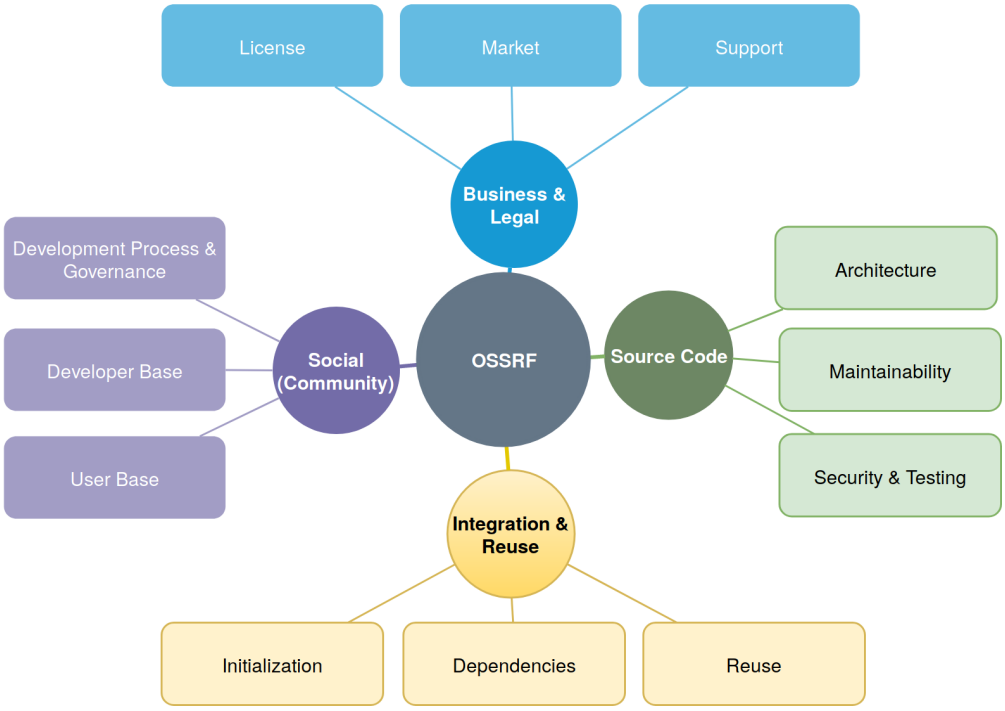


Figure 4.1: Open Source Software Resilience Framework (OSSRF): Dimensions & Goals

Source Code	Architecture (G01)	Robustness (I01)
		Scalability (I02)
		Usability (I03)
		Effectiveness (I04)
	Maintainability (G02)	Corrections (I05)
		Improvements (I06)
	Security & Testing(G03)	Security (I07)
		Testing process (I08)
		Coverage (I09)

Figure 4.2: Source Code Dimension, Goals & Indicators

- **Architecture (G01):** this goal is related to the aspects of the source code that structurally strengthen the project and promote functionality and scaling. We propose this goal in alliance with the “Minimal human vulnerability” goal that can be found in CRF which promotes, as more resilient, cities with infrastructures that provide stability, effectiveness, sanitation and robustness, access to energy supply or drinking water.

- **Maintainability (G02):** this goal relates to the maintainability of source code. OSS projects, being collaboratively and voluntarily evolving projects, often need to go through phases of refactoring, correction or undergo necessary improvements. Especially after a crisis (for example a competitive open source solution wins the at-

tention of the majority of the contributors' base) an OSS project needs to regroup as soon as possible. In the CRF we find the "Diverse livelihoods and employment" goal which similarly aims at maintaining the social capital after a shock (for example with supportive financing mechanisms) and improve or correct by training and promoting business development and innovation.

- **Security & Testing (G03):** this goal is related to the aspects that promote the security and correctness of the OSS project. As in "Effective safeguards to human health and life", a goal found in CRF, this goal is about foundational structures that ensure a tested and fully functioning system.

4.1.2 Indicators

The goals are further decomposed to the following indicators (we base our definitions mainly in Miguel et. al. literature review [75]. In case an indicator's definition is not based on the aforementioned work, we will be providing references accordingly):

- **Robustness (I01):** is defined as "the degree to which an executable work product continues to function properly under abnormal conditions or circumstances". We propose this as a qualitative indicator (Likert Scale) described with the following values. 1 - limited, 2 - little, 3 - moderate, 4 - good, 5 - great.
- **Scalability (I02):** is defined as "the ease with which an application or component can be modified to expand its existing capabilities. It includes the ability to accommodate major volumes of data". We propose that this should be a qualitative indicator (Likert Scale) described with the following values. 1 - limited, 2 - little, 3 - moderate, 4 - good, 5 - great.
- **Usability (I03):** is defined as "the degree to which the software product makes it easy for users to operate and control it". We propose that this should be a qualitative indicator (Likert Scale) described with the following values. 1 - limited, 2 - little, 3 - moderate, 4 - good, 5 - great.

  When the OSSRF is being applied, the aforementioned indicators (robustness, scalability and usability) should be provided as qualitative variables, by an expert. We base this decision in [98] where Wasserman treats those indicators the same way in OSSpal.
- **Effectiveness (I04):** is defined as the percentage of critical bugs fixed the last six months to all bugs fixed in the last six months. This indicator derives from the SQO-OSS quality model as published in [85].

  At this point we would like to clarify that the Effectiveness indicator can follow the definition if the OSS project's issue tracker offers a category for the critical bugs, separating them from the rest. In any other case this indicator could be treated as a qualitative (Likert Scale) with the following values. 1 - limited, 2 - little, 3 - moderate, 4 - good, 5 - great.

- **Corrections (I05):** is proposed as part of the maintainability goal to “try and capture the degree to which the software can be modified to serve correction purposes”.

- **Improvements (I06):** is proposed as part of the maintainability goal to “try and capture the degree to which the software can be modified to serve improvement purposes”.

  Since both (corrections and improvements) are indicators that can apply to several different aspects of the software (i.e. changes of the environment of the software, the requirements, the functional specification) as Miguel states in [75], we propose them as qualitative indicators (Likert Scale) with the following values. 1 - limited, 2 - little, 3 - moderate, 4 - good, 5 - great.

- **Security (I07):** is defined as “the protection of system items from accidental or malicious access, use, modification, destruction, or disclosure”. We propose that this should be a qualitative indicator (Likert Scale) with the following values. 1 - limited, 2 - little, 3 - moderate, 4 - good, 5 - great. We base our choice for this indicator in [98] where Wasserman treats this indicator the same way in OSSpal.

- **Testing process (I08):** is proposed as a boolean indicator to verify that the OSS project under assessment follows a typical process as far as the testing is concerned (i.e. unit testing, test driven design techniques). In [64] the authors provide an empirical study that shows the importance of test driven techniques in software development.

- **Coverage (I09):** is defined as “the ratio of basic code blocks that were exercised by some test, to the total number of code blocks in the system under test” [32]. Therefore this is proposed as a percentage indicator [0, 100%].

4.2 The Business & Legal Dimension (D02)

The Business & Legal dimension for an OSS project has become extremely important nowadays as it provides the legal framework under which an OSS project can be used, reused and even commercialized. OSS projects are mainly dependent in voluntary work but, as the authors of [80] argue, we often see more mature projects utilizing Open Source Business Models to offer commercial services. The equivalent dimension of CRF, Economy & Society is related with organization focusing, as well, to the aspects of sustainable economy, rule of law and community support. In Figure 4.3 we can see a detailed analysis of the dimension including the relevant goals and indicators.

4.2.1 Goals

We are proposing the following goals for the Business & Legal dimension:

Business & Legal	License (G04)	License type (I10)
	Market (G05)	Dual licensing (I11)
		Commercial resources (I12)
		Commercial training (I13)
		Industry adoption (I14)
	Support (G06)	Non profit / Foundation support (I15)
		For profit company support (I16)
		Donations (I17)

Figure 4.3: Business & Legal Dimension, Goals & Indicators

- **License (G04):** this goal aims to investigate the legal aspects of an OSS project. In alliance with the goal in CRF "Comprehensive security & rule of law" this goal describes the legal framework under which the OSS project is published in order to pro-actively secure its openness and availability to be used, reused and shared according to the license terms.

- **Market (G05):** this goal is proposed in alliance with the "Sustainable economy" goal of the CRF that takes under consideration the aspects of the business environment, the diverse economic base and business continuity planning. Following this paradigm, in OSS we respectively study the aspects related to market and commercial use of an OSS project. Red Hat, Inc., for example is a well-known company that uses a dual-licensing model for some of its products. The company provides enterprise-level support and services for open-source software, and it uses a dual-licensing model to provide both open-source and commercial licenses for its products. Red Hat's most well-known product is Red Hat Enterprise Linux (RHEL), which is an open-source operating system based on the Linux kernel. RHEL is available under a dual-licensing model, with a free, open-source version that is licensed under the GPL and a commercial version that is licensed under a proprietary license. Customers who use the commercial version of RHEL receive access to enterprise-level support, security updates, and other services, while users of the open-source version can access the source code and make modifications. MongoDB is another popular, cross-platform document-oriented database system that uses a dual-licensing model. The software is available under the Server Side Public License (SSPL), an open-source license that was introduced by MongoDB, and a commercial license. Under the SSPL, users are free to use, modify, and distribute the software for any purpose, as long as they comply with the license's terms. The commercial license, on the other hand, provides customers with additional features, support, and services.

- **Support (G06):** this goal is related to a rather controversial subject in OSS. In [42], Daffara refers to the myth that OSS "is not reliable or supported" and argues against it. With the adoption of OSS software in vital parts of companies and organizations

(i.e. web servers running Apache and Linux) it has become evident that professional support is key for an OSS project to become a success. Support helps the end user to feel safe (i.e. when crisis strikes or during a shock) and provides a sense of belonging. Same goes for the “Collective identity and community support” goal that we find in CRF, referring to the beneficial role of collective identity and local community support, especially in times of crisis.

4.2.2 Indicators

We are further decomposing the goals to the following indicators:

- **License type (I10):** Using a license for an OSS project along with the type of this license play a significant role in the evolution and success of OSS. In [94] the authors study how licenses, depending on the level of permissiveness (i.e. copyright versus copyleft) or the level of persistence (GPL versus LGPL) can affect the OSS project in terms of adoption and commercialization. In [63] Lindman et. al. argue that licensing can often be a complex task for OSS teams and that is why we find structured license selection processes mainly in big OSS projects. Taking the above under consideration, we propose the following values for this specific indicator: 1 - all restrictive, 2 - not licensed, 3 - mixed license, 4 - persistent license (i.e. GPL), 5 - all permissive license (i.e. MIT).
- **Dual licensing (I11):** Whereas dual licensing is not necessarily a success factor for an OSS project, based on the literature, studies like [95] and [42] argue that dual licensing is a key factor when it comes to commercialization of an OSS project. Therefore, we consider it a plus, when it comes to the market goal. We propose this indicator as boolean: 0 - for non dual licensed projects and 1 - for dual licensed ones.
- **Commercial resources (I12):** Providing commercial resources (i.e. user guides or merchandise) is a known business model for OSS projects. We propose that this indicator is boolean with: 0 - for projects with no commercial resources and 1 - for project with commercial resources.
- **Commercial training (I13):** Providing commercial training (i.e. video tutorials or Massive Open Online Courses (MOOCs)) is another known business model for OSS projects. We propose that this indicator is boolean with: 0 - for projects with no commercial resources and 1 - for project with commercial resources.

  In regard with commercial resources and commercial training in [77] the authors study how well known companies like IBM and Redhat, achieved competitiveness and economic growth by providing added value services to their open source solutions.
- **Industry adoption (I14):** An OSS project that manages to attract the interest of the industry is more likely to become successful in the market. In [42] the authors

argue that OSS boosts both innovation and software development speed whereas in [84] a scientific work about open innovation and the SME food industry authors highlight that “open innovation offers SMEs a special avenue to better compete in the marketplace”. We propose this indicator as boolean with: 0 - indicating projects with no industry adoption and 1 - indicating projects that have been adopted by the industry.

- **Non profit / Foundation support (I15):** Many successful OSS projects, are supported by non profit organizations. Some times these NGO are created to support specifically an OSS project (i.e. Free Software Foundation, Linux Foundation, WordPress Foundation, Blender Foundation and so forth) as mentioned in [56]. We define this indicator as boolean with 0 - for projects not supported by a non profit organization and 1 - for projects supported by a non profit organization.
- **For profit / company support (I16):** As with the Non profit / Foundation support indicator, the “For profit / company support” indicator takes under consideration the existence of a company “attached” or supporting an OSS software. There are examples from well known projects that have helped companies built business models around them (i.e. Redhat offering paid services for Linux installations or Automattic for WordPress). We propose this indicator as boolean with 0 - for projects not supported by a company and 1 - for projects supported by a company.
- **Donations (I17):** Donations, have been one of the most known ways for OSS projects to earn money, since the early days of open source software. In [58] the authors refer to donations as “indicator of acceptance”. We propose that this indicator is boolean with 0 - for projects not accepting donations and 1 - for projects that accept donations.

4.3 The Integration & Reuse Dimension (D03)

The third dimension of CRF, related to place, was designed to study the connection and communication of the city’s ecosystems. We designed OSSRF’s respective dimension to study the levels of integration and reuse of the OSS project. Since OSS projects usually reuse components of other OSS projects or are being reused themselves it is critical to be able to measure the performance of a project on this aspect. In Figure 4.4 we can see a detailed analysis of the dimension including the relevant goals and indicators.

4.3.1 Goals

We are proposing the following goals for the Integration & Reuse dimension:

- **Initialization (G07):** this goal is proposed having in mind the ability of the project to be initialized in such a way that it will help its uninterrupted functionality. The

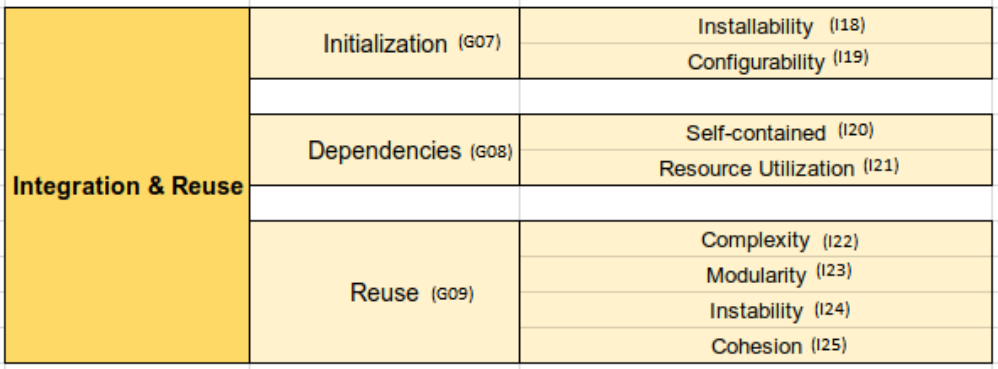


Figure 4.4: Integration & Reuse Dimension, Goals & Indicators

same goal is also relative to the agility of an OSS regarding its configuration. We argue that this goal shares conceptual similarities with the "Effective provision of critical services" goal of the CRF that takes under consideration all those factors that predefine and protect critical assets, services and ecosystems within a city.

- **Dependencies (G08):** this goal takes under consideration the dependencies that an OSS project uses in order to function properly. Dependencies are as good for the project as their quality and resilience. This is why this goal, is aligned with the "Reduced exposure and fragility" goal of the CRF.
- **Reuse (G09):** this goal is about the ability of the OSS project, in a whole or at a component level, to be reused by other OSS projects. Reusability of a project, apart from making it a good candidate to complete another software's requirements is also an indicator of high quality architecture and source code. In our framework this goal aligns with the "Reliable mobility and communications" goal of the CRF model in the sense that it reusable components promote mobility and tend to integrate or be integrated easily with other OSS projects.

4.3.2 Indicators

We are further decomposing the goals to the following indicators (we base our definitions mainly in Miguel et. al. literature review [75]. In case an indicator's definition is not based on the work, it will be followed by a separate reference to the respective source):

- **Installability (I18):** is defined as "the degree to which the software product can be successfully installed and uninstalled in a specified environment". We propose that this should be a qualitative indicator described with the following values provided by an expert. 1 - limited, 2 - little, 3 - moderate, 4 - good, 5 - great.

  We base our choice for the aforementioned indicator, installability, to be qualitative (Likert scale) in [98] where Wasserman treats those indicators the same way in OSSpal.

- **Configurability (I19):** is defined as “the ability of the component to be configurable”. In [72] the authors argue that highly-configurable systems lead to exponentially growing configuration spaces making quality assurance challenging. Based on that we propose that this should be a qualitative indicator described with the following values provided by an expert. 1 - limited, 2 - little, 3 - moderate, 4 - good, 5 - great.
- **Self-contained (I20):** is defined as “the function that the component performs must be fully performed within itself”. In [71] the authors conduct a performance evaluation of open source graph databases projects and conclude that self-containment makes the project a better candidate over competitive ones. To our best knowledge, there is not a well established metric for the self-containment of an OSS project we propose that this should be a qualitative indicator described with the following values provided by an expert. 1 - limited, 2 - little, 3 - moderate, 4 - good, 5 - great.
- **Resource Utilization (I21):** is defined as “the degree to which the software product uses appropriate amounts and types of resources when the software performs its function under stated conditions”. In [89] the authors study operating systems and highlight that often, an OSS project is designed with the end user in mind and thus the focus is mainly ease of use or performance and security and not resource utilization. This difficulty in having a clear metric on resource utilization led as to propose that this should be a qualitative indicator described with the following values provided by an expert. 1 - limited, 2 - little, 3 - moderate, 4 - good, 5 - great.
- **Complexity (I22):** or CC is defined as “a quantitative measure of the number of linearly independent paths through a program’s source code” by McCabe [70]. The authors of [36] and [99] provide the following groups of values for the CC metric, as seen on Table 4.1.

Table 4.1: Complexity (CC) McCabe’s grouped values

Value range	Qualitative analysis
0 - 15	simple program, without much risk
16 - 20	moderate complexity, moderate risk
21 - 50	high complexity, high risk
$>$ 50	untestable, very high risk

In the studies there’s a debate on whether the first group of values should stop at 10 or 15 describing the first group as without much risk and the second group of moderate risk. We propose an indicator that provides a 5th tier of complexity, taking under consideration the threshold of 15, as seen on Table 4.2:

Without loss of generality, the originally proposed tier of 0-15, described as “simple program, without much risk” is further decomposed to trivial program with not much risk and simple program with little risk.

So depending on McCabe’s CC metric this indicator’s values are described as follows based on the risk deriving from the complexity of the product: 1 - very high risk, 2 - high risk, 3 - moderate risk, 4 - little risk, 5 - not much risk.

Table 4.2: Complexity (I22) OSSRF metric's proposed values

Value range	Qualitative analysis	OSSRF value
0 - 10	simple program, without much risk	5
11 - 15	little complexity, little risk	4
16 - 20	moderate complexity, moderate risk	3
21 - 50	high complexity, high risk	2
$>$ 50	untestable, very high risk	1

- **Modularity (I23):** is defined as "the degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components". As Viseur states in [96] high modularity of an OSS project is a competitive advantage for developers and, at the same time, allows users to gradually discover and use functionality (i.e. Mozilla Firefox add-ons). To our best knowledge, there is not a well established metric for assessing the modularity of an OSS project therefore we propose that this should be a qualitative indicator described with the following values (Likert scale) provided by an expert. 1 - limited, 2 - little, 3 - moderate, 4 - good, 5 - great.

- **Instability (I24):** is defined by Martin [66] as "I = (Ce % (Ca+Ce))", that is the ratio between efferent coupling (ce) and afferent coupling (ca). This metric has a range [0,1] where I = 0 indicates a maximally stable category and I = 1 indicates a maximally instable category. The lowest the number the more stable the project therefore for this indicator the final value that we use to the framework calculations is 1 - I.

- **Cohesion (I25):** is measured by Cidamber and & Kemerer in [40] using the Lack of cohesion in methods (LCOM). We then use the thresholds provided by Ferreira et al in [47] to evaluate the resulting value of LCOM , based on the size of the software, as shown in Table 4.3:

Table 4.3: Thresholds as proposed by Ferreira et al [47] to evaluate the resulting value of LCOM, based on the size of the software

Size (number of classes)	LCOM (good / regular / bad)
$<=$100	0 / 1-25 / $>$25
101-1000	0 / 1-20 / $>$20
$>$1000	0 / 1-20 / $>$20

Therefore our cohesion indicator ranks between [1,3] with 1 - indicating bad cohesion, 2 - regular cohesion and 3 - good cohesion.

At this point we would like to clarify that the Instability and Cohesion indicators can follow the definitions only if the OSS project follows the object oriented development style. If the OSS project is not object oriented we propose that the indicators should be considered qualitative indicators described with the following (Likert scale) values provided by an expert. 1 - limited, 2 - little, 3 - moderate, 4 - good, 5 - great.

4.4 The Social (Community) Dimension (D04)

The last dimension of CRF is about Leadership & Strategy. In OSS projects there is extensive work for the community that is being built around an OSS software. Leadership, strategy and knowledge acquisition usually derive from an OSS project's community (i.e. feature proposal, bug reports, translations, documentation or testing). In the Social (Community) Dimension we are proposing goals and indicators regarding the development process, the governance, the development and user base with the aim of studying how strong the social capital of the OSS project is. In Figure 4.5 we can see a detailed analysis of the dimension including the relevant goals and indicators.

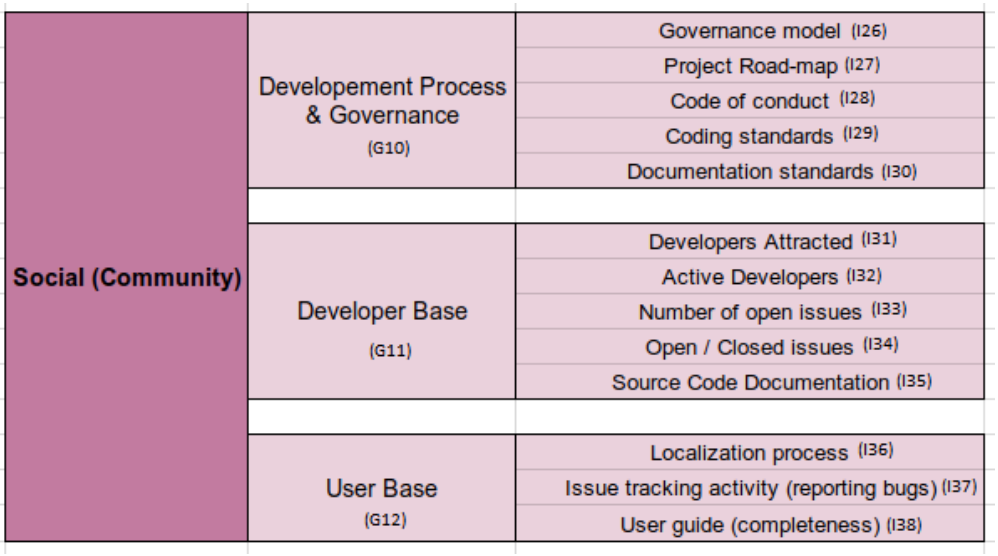


Figure 4.5: Social (Community) Dimension, Goals & Indicators

4.4.1 Goals

We are proposing the following goals for the Social (Community) Dimension:

- **Developement Process & Governance (G10):** this goal was designed and fully aligns with the "Effective leadership and management" of the CRF. Its indicators verify that the project has all the necessary information to guide its users and developers through the process of evolving the software. It also provides the necessary mechanisms to ensure a friendly and open environment that everyone can contribute on the project in the context of a governance model. We suggest that this specific indicator takes the value 1 if the governance model that it is used is one of the state of the art OSS governance models as seen in [26].
- **Developer Base (G11):** this goal is related with the development community of the project. In order for an OSS project to be successful this part of the community needs

to always stay motivated and active. This goal is similar to “Integrated development planning” of the CRF.

- **User Base (G12):** this goal is related with the end users of the OSS project. It can potentially include members of the development community that are also users of the software or the have undertaken the role of a tester. This part of the community are the “customers” of the OSS project and hence it is really important to be keep the engaged and motivated as their feedback (i.e. feature proposals, bug reports and so forth is invaluable). This goal is aligned with the “Empowered stakeholders” of CRF.

4.4.2 Indicators

We are further decomposing the goals to the following indicators:

- **Governance model (I26):** the existence of a governance model for an OSS project is considered mandatory, especially if it wants to become self sustainable using one or more of the business models discussed in the business and legal dimension. We propose this indicator as boolean with: 0 - for projects that do not utilize a governance model and 1 - for project that utilize governance models.

- **Project Road-map (I27):** the project roadmap, as with the Governance model, is an indicator of a well organized project with clear goals and milestones that wants to clearly share with its community. We propose this indicator as boolean with: 0 - for projects that do not use roadmaps and 1 - for projects that use roadmaps.

  In [65] the authors study community aspects for well known, hybrid, OSS projects with commercial success and highlight both governance and roadmap existence as indicators of healthy OSS communities.

- **Code of conduct (I28):** the fact that OSS projects form global, diverse communities that work asynchronously need to set the rules of communication and interaction between their members. We propose this indicator as boolean with: 0 - for projects that do not use a code of conduct and 1 - for projects that use one.

- **Documentation standards (I29):** another critical indicator for the success of community driven projects, as are OSS projects is standards for the documentation of the source code. This helps newcomers to easily understand the existing code base and smoothly become a part of the team. We propose this indicator as boolean with: 0 - for projects that do not follow a documetation standards and 1 - for projects that follow one.

  In [37] the authors investigate work practices used by contributors to well established OSS projects and highlight the use of both Code of conduct and Documentation standards.

- **Coding standards (I30):** coding standards have been always a part of the OSS projects documentation. They serve as the source code development manual for the developers in the community of the OSS and they have been adopted by the leaders of the free / open source software movement (Linux Kernel, GNU, and so forth). Coding standards indicate professionalism and maturity for the OSS project. We propose this indicator as boolean with: 0 - for projects that do not use coding standards and 1 - for projects that use coding standards.

  The following indicators were inspired mainly by the works of Robles et al [83] and Wasserman [98]. We will be providing respective references per indicator, where necessary.

- **Developers Attracted (I31):** is proposed as the rate of developers joined the project in the last six (6) months to the total number of developers. The indicator's value ranks between [0,1].

- **Active Developers (I32):** is proposed as the rate of developers that have been active, contributing to the project, the last six (6) months to the total number of developers. Depending on the version control system (CVS) the project uses (i.e. Trac, Git, Mercurial), active developers are the ones that have contributed commits in the CVS in the timeframes defined above. The indicator's value ranks between [0,1]."

- **Number of open issues (I33):** is proposed as the number of the current open issues to the total issues reported since the beginning of the project. This indicator gives us a perspective of the activity of the community regarding bug reporting. This indicator ranks between [0,1]. The lowest the number the less open issues is has therefore for this indicator the final value that we use to the framework calculations is 1 - Number of open issues.

- **Open vs Closed issues (I34):** is proposed as the number of issues closed in the past six (6) months to the number of issues opened in the past six (6) months. This indicator gives us a perspective of the activity of the community regarding bug fixing. This indicator ranks between [0,1].

- **Source Code Documentation (I35):** is defined as the rate between the number of comment lines of code (CLOC) to the number of the lines of code (LOC). This indicator gives us a perspective of the documentation effort done regarding to the source code. This indicator ranks between [0,1].

- **Localization Process (I36):** OSS project localization process (i.e. translation of the software and / or project resources) is a best practice that is backed up by literature. In [90] the authors argue that software translations benefit the evolution and growth of OSS and thus, should be one of the project leaders priorities. We propose that this indicator has the following values: 0 - no localization process is defined, 1 - localization process is defined.

- **Issue tracking activity (reporting bugs) (I37):** is defined as the number of bugs reported the past six (6) months to the number of reported bugs since the beginning of the software. This indicator ranks between [0,1].

  We would like to clarify that the co existence of bug reports and open / closed issues serves the need to separately assess the bug reports that come from the end users to the technical issues that are usually reported by the developers of the project. We acknowledge that sometimes the developers of an OSS project also act as end users but, as the authors of [54] state, often times end users' reports are misclassified as bugs when they are really features (i.e. code enhancement requests or customization requests).

- **User guide (completeness) (I38):** this indicators has the goal of evaluating the maturity of an OSS project's user guide. User guides have been adopted by the most evolved and well known OSS projects (for example GNU Emacs user guides [14]). We propose the indicator's values as follows: 1 - non existing user guide, 2 - on hiatus / discontinued, 3 - pre release (alpha / beta / release candidate), 4 - released (version 1.0+), 5 - commercial versions of the guide.

4.5 Resilience determination mechanism

Since the assessment of a project regarding its resilience is based on indicators we need a mechanism to determine whether the OSS project under review is resilient and, how its resiliency changes as the project evolves over time. Starting from the indicators' level we will consider an OSS project successful towards a resilience goal when at least 50% of the goals indicators are considered resilience.

Moving to the dimensions level, an OSS project will be considered resilient towards a dimension when at least to 50% of the goals of this specific dimension are considered resilient. Finally, overall, a project will be considered resilient, when at least two (2) out of the four (4) dimensions (50%) of are considered resilient.

To assist this mechanism we express all the values in the indicators level to percentages. More specifically the framework has four (4) types of indicators. Boolean indicators, Likert scale indicators, Indicators ranking between [0,1] and percentage indicators. In order to simplify the visualization of decisions and results, for starters we express the indicators' level to percentages. For boolean indicators this means that the 0 / 1 values are transformed to 0% / 100%. For the Likert scale indicators, depending on the number of available options (either 3 or 5 in our case) the value is being divided by the total of possible answers (i.e. on an Likert scale indicator with 5 possible values, if we have a score of three this will be expressed as 3 / 5 = 60%). For the percentage indicators and the indicators with values between [0,1] we don't need to do any transformation as they are already expressed as percentages.

On the next level of the framework, the goals level, we are calculating the average of the indicators for each goal and the resulting percentage is the value of the goal. Finally

for the upper level, the dimensions level we are following the same process as with the goals. We are calculating the average of the goals for each dimension and the resulting percentage is the value of the dimension.

At this point we would like to point out that the proposed framework considers all the the indicators, goals and dimensions equal regarding their importance on the resilience framework (there are no weights). This decision alongside with the aforementioned 50% threshold are reported as threats to validity and it is in our future goals to study and try to approach them empirically.

4.6 Applying Resilience Framework on Open Source Software: Indicators &Tools

In this section we are applying the OSSRF to six (6) OSS projects with the aim of providing a proof of concept that OSSRF can distinguish projects that are intuitively resilient from projects that are intuitively non resilient. From the selected projects three (3) are intuitively resilient and three (3) are intuitively non resilient. In order to present the concept of an OSS project's resilience evolution over time, for each one of the aforementioned projects (resilient and non resilient) we will be assessing their resilience using OSSRF for a number of major, consecutive releases.

In the next part we are going to analyze the application of the framework, providing, where necessary, the tools and information used for the proposed frameworks application. For brevity, we will not be presenting the numbers for all thirty eight (38) indicators for all six (6) projects. Instead we are presenting the results on the goals and dimensions levels. The readers of the work are encouraged to find the raw data to `https://doi.org/10.5281/zenodo.5576580`.

4.6.1 Indicator types

4.6.1.1 Qualitative indicators

The following indicators are qualitative and are evaluated by an expert: Robustness, Scalability, Usability, Corrections, Improvements, Security, Installability, Configurability, Dependability, Resource Utilization, Modularity.

In the absence of an expert and in order to keep the experiment as unbiased as possible we will be using average values (3) for the aforementioned indicators for the resilient group of projects expecting that the non average values will highlight the resilience of the project. For the non resilient projects, we will adopt the value of (2) for the qualitative indicators. The reason we will be doing that is that, percentage wise, the qualitative factor give on average a 60% score to each indicator boosting the average above 50%. Since most of the non resilient project have a lifespan of 2 years and little activity and contributors community we believe that, without loss of generality, we can inject a small penalty to qualitative indicators such as robustness, scalability, usability and so forth.

To verify our decision we conducted interviews with 5 experts. The background of the experts is as follows:

1. **Expert #1:** has working experience as an OSS practitioner (software engineering) and researcher for more than 10 years. Expert #1 holds a masters degree in Computer Science.
2. **Expert #2:** has working experience as a researcher in Computer Science for more than 3 years. Expert #2 holds a masters degree in Computer Science.
3. **Expert #3:** has working experience as an OSS practitioner (software engineering) for more than 5 years. Expert #3 holds a masters degree in Computer Science.
4. **Expert #4:** has working experience as an OSS practitioner for more than 5 years and as a researcher for more than 2 years. Expert #4 is a PhD Candidate in the field of Computer Science.
5. **Expert #5:** has working experience as a researcher for more than 15 years. Expert #5 is a Post Doc in Computer Science.

We presented the 6 projects as seen in Section 7 to the experts (identifying them as resilient and non resilient which is exactly the way we ran our tests for this scientific work) and we presented them with the definition of resilience as adopted from the CRF for the purposes of this manuscript. We also presented to them the qualitative indicators (and their definitions) as defined in this work. Then we asked them to independently provide, in their expert opinion, the appropriate values for the qualitative indicators (scoring them from 1 to 5, following the Likert scale).

For the non resilient projects, for all the indicators, we have an average score of 2 from our experts, with the exception of the Scalability Indicator (I02) that scored an average of 1. This validates that for the qualitative indicators it was reasonable to inject the penatly we chose. For the resilient projects, for all the indicators, we have an average score of 4 from our experts, with the exception of Security (I07) that got an average of 5. This validates that using the median value (3) in our tests was more conservative than an expert would probably do.

For brevity reasons we chose not to share the raw data of the aforementioned analysis in the manuscript. The readers of the work are encouraged to find the raw data to `https://doi.org/10.5281/zenodo.5576580`.

4.6.1.2 Mixed indicators

There are also mixed indicators. Complexity, Instability, Cohesion will be measured, for object oriented projects whereas for non object oriented projects, these indicators will be considered qualitative and will be treated as described in the qualitative indicators section. Effectiveness is also a mixed indicator. If the project's issue tracker provides

categorization for the critical bugs then the indicator can be measured. On a different case it will be treated as described in the qualitative indicators section taking an average value which, since it is a percentage indicator will be 50%.

4.6.1.3 Sensitivity & Veto Principles investigation

Our assessment model in its current version is unweighted, which means that all the indicators are equally contributing to the decision on whether an assessment concludes to a resilient or non resilient result. This and the fact that 14 of our models indicators are boolean lead to a concern that a specific value of a specific indicators (in the absence of weights) might be able to independently impact the decision of our assessment model.

To address that concern we have conducted one-factor-at-a-time sensitivity analysis. More specifically we experimented by applying changes to one indicator at a time keeping all the other indicators of the model to their baseline values. This analysis led to the following discoveries:

1. **Factors with high sensitivity:** the only factor that presents high sensitivity is Testing Process (I08). More specifically if this boolean factor get the value 1 (true), it significantly increasing the resilience score for Source Code Dimension (D01). We have added this finding to our limitations and threats to validity section.
2. **There are no indicators to our model that function as veto principles:** apart from the one-factor-at-a-time sensitivity analysis with baseline values, we repeated the analysis on a set of indicators values that lead to a resilient and non resilient project respectively. This way we wanted to ensure that a single indicator cannot independendly alter the result of our model assessing a non resilient project as resilient and vice versa.

For brevity reasons we chose not to share the raw data of the aforementioned analysis in the manuscript.

4.6.2 Tools used for indicator measurement

In order to be able to apply the OSSRF to the selected projects the following tools were used:

1. **OSS Project's official website:** it was used to provide information for the following indicators: all the Market related indicators, all the Support related indicators, coding standards and documentation standards indicators and the user guide indicator.
2. **OSS Code Repositories (i.e. Github):** it was used to provide information for the following indicators: Effectiveness (if the source code repository's issue tracker is

being used by the project), Testing process, License, governance model, project road-map and code of conduct indicators, all of the Developer base related indicators (if the source code repository's issue tracker is being used by the project) and the issue tracking activity indicator.

3. **PHPCoverage Tool [21]:** This open source tool was used to measure the coverage indicator for object oriented projects written in PHP.

4. **PHPQA [25]:** This open source tool was used to measure the following indicators for object oriented projects written in PHP: Complexity, Instability, Cohesion, Documentation.

4.7 Applying Resilience Framework on Open Source Software: Resilient and Non Resilient Projects

In this section we will be sharing the application of the OSSRF assessment model to 5 concecutive versions of 3 intuitively resilient and 3 intuitively non resilient projects. We selected these projects in order to present that the model discussed in this manuscript can successfully distinguish between resilient and non resilient projects while they evolve in time (hence the 5 consecutive versions).

First we will provide a brief description of the project sharing some context that will present our reasoning for choosing it as intuitively resilient or no resilient. Following you will find the scores of the goal and dimension levels for all the versions to which we applied the OSSRF. To better assist the interpretation of the results we will be also sharing charts showing the trend of the four dimensions of OSSRF as each project evolves from one version to the next. For clarity we present the six projects and their respective versions to the following table.

OSS Project	Versions assessed	Resilient / Non Resilient
Laravel	5.4, 5.5, 5.6, 5.7, 5.8	Intuitively Resilient
Composer	1.4.0, 1.5.0, 1.6.0, 1.7.0, 1.8.0	Intuitively Resilient
PHPMyAdmin	4.4, 4.5, 4.6, 4.7, 4.8	Intuitively Resilient
OKApi	1.1, 1.2	Intuitively Non Resilient
PatternalPHP	0.3, 0.6, 0.7, 1.0, 2.0	Intuitively Non Resilient
PHPExcel	1.7.8, 1.7.9, 1.8.0, 1.8.1, 1.8.2	Intuitively Non Resilient

4.7.1 Resilient projects

In the section we present the application of the OSSRF to three (3) projects we intuitively classify as resilient.

4.7.1.1 Laravel

Laravel [16] is a well known open source project in the domain of PHP web frameworks. According to this report [1] was the top solution for 2021. Laravel was first released in 2011 (10 years lifespan) and according to Github [17], has hundreds of contributors, is watched by more than 4,000 users and was forked more than 22,000 times as of the time of writing. On the following table you can see the values of the OSSRF analysis for each version.

	v5.4	v5.5	v5.6	v5.7	v5.8
D01 - Source Code (%)	67	68	67	67	66
G01 - Architecture (%)	57	57	57	57	57
G02 - Maintainability (%)	60	60	60	60	60
G03 - Security & Testing (%)	84	85	84	82	82
D02 - Business (%) & Legal	81	81	81	81	81
G04 - License (%)	100	100	100	100	100
G05 - Market (%)	75	75	75	75	75
G06 - Support (%)	67	67	67	67	67
D03 - Integration (%) & Reuse	63	63	62	63	63
G07 - Initialization (%)	60	60	60	60	60
G08 - Dependencies (%)	60	60	60	60	60
G09 - Reuse (%)	69	68	68	68	68
D04 -Social (Community)	71	70	70	69	68
G10 - Dev. Process (%) & Governance	80	80	80	80	80
G11 - Developers Base (%)	60	60	61	58	57
G12 - User Base (%)	72	70	69	69	68

On Figure 4.6, we can see that all four dimensions' score above 60% for all consecutive major releases. Therefore, following the resilience determination mechanism of our framework the project is considered resilient.

4.7.1.2 Composer

Composer [4] is a dependency manager for the PHP programming language. It has been evolving for 10 years now, has been forked nearly 6,500 times and has a community of nearly 1,000 developers on Github [5]. On the following table you can see the values of the OSSRF analysis for each version.

On Figure 4.7, we can see that all four dimensions' score above 50% for all consecutive major releases. Therefore, following the resilience determination mechanism of our framework the project is considered resilient. It is worth mentioning that we see a slow decline over the last versions in the Reuse & Integration and Social dimensions. This should alert the project leaders to possible structural and community base stressors that the project might be facing.

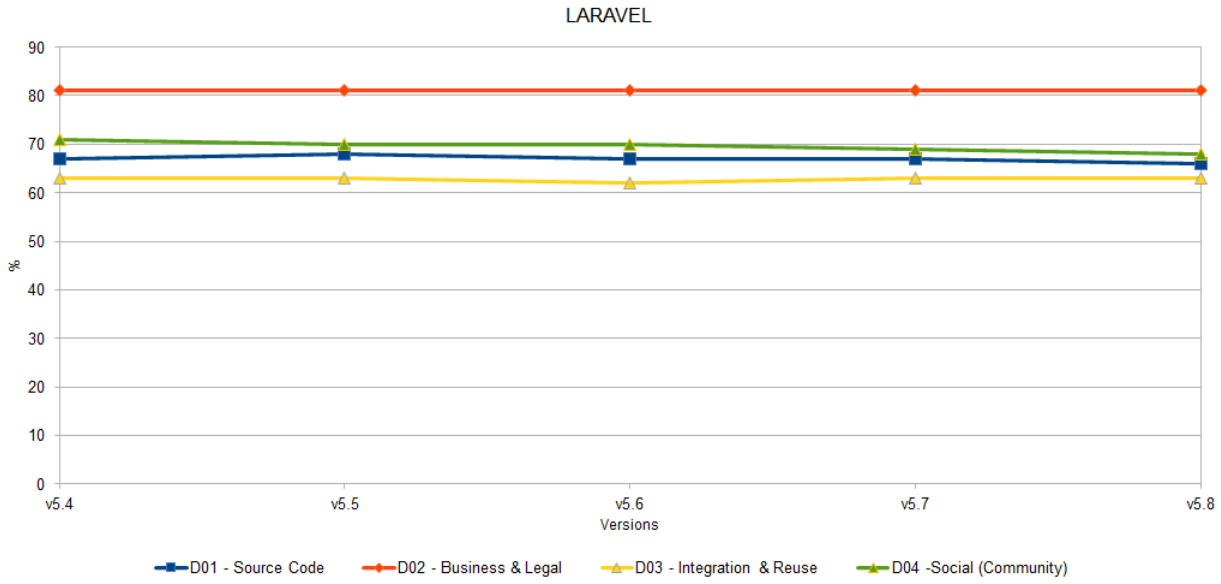


Figure 4.6: Resilience evolution between releases for Laravel in OSSRF dimensions level

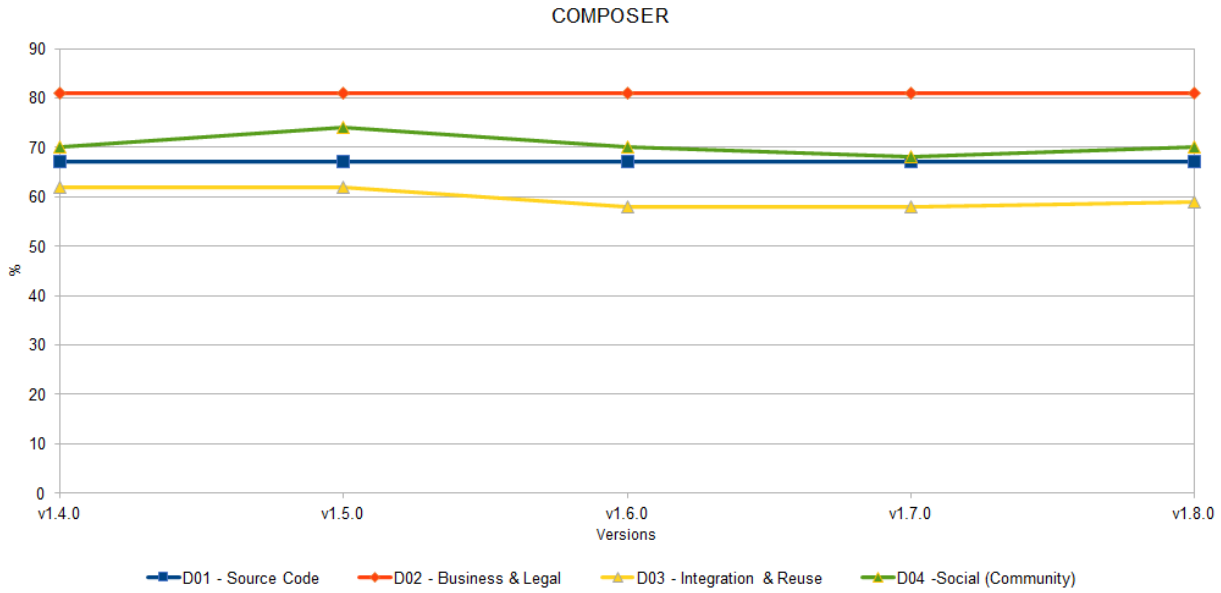


Figure 4.7: Resilience evolution between releases for Composer in OSSRF dimensions level

4.7.1.3 PHPMyAdmin

PHPMyAdmin [23] is a free database management system written in PHP language with the aim of managing MySQL databases over the web. It was established as a project in 1998 which means a lifespan of 23 years. On Github [24] we see it has been forked 3,200 times and has more than 1100 developers working on its community. On the following

Table 4.4: OSSRF assessment for the Composer project

	v1.4.0	v1.5.0	v1.6.0	v1.7.0	v1.8.0
D01 - Source Code (%)	67	67	67	67	67
G01 - Architecture (%)	57	57	57	57	57
G02 - Maintainability (%)	60	60	60	60	60
G03 - Security & Testing (%)	83	84	83	83	83
D02 - Business (%) & Legal	81	81	81	81	81
G04 - License (%)	100	100	100	100	100
G05 - Market (%)	75	75	75	75	75
G06 - Support (%)	67	67	67	67	67
D03 - Integration (%) & Reuse	62	62	58	58	59
G07 - Initialization (%)	60	60	60	60	60
G08 - Dependencies (%)	60	60	60	60	60
G09 - Reuse (%)	65	65	54	54	56
D04 -Social (Community) (%)	70	74	70	68	70
G10 - Dev. Process (%) & Governance	80	80	80	80	80
G11 - Developers Base (%)	63	72	60	57	62
G12 - User Base (%)	68	68	69	68	68

table you can see the values of the OSSRF analysis for each version.

Table 4.5: OSSRF assessment for the PHPMyAdmin project

	v4.4	v4.5	v4.6	v4.7	v4.8
D01 - Source Code (%)	66	67	66	66	67
G01 - Architecture (%)	57	57	57	57	57
G02 - Maintainability (%)	60	60	60	60	60
G03 - Security & Testing (%)	80	83	81	81	83
D02 - Business (%) & Legal	81	81	81	81	81
G04 - License (%)	100	100	100	100	100
G05 - Market (%)	75	75	75	75	75
G06 - Support (%)	67	67	67	67	67
D03 - Integration (%) & Reuse	60	58	58	60	60
G07 - Initialization (%)	60	60	60	60	60
G08 - Dependencies (%)	60	60	60	60	60
G09 - Reuse (%)	60	54	53	64	58
D04 -Social (Community) (%)	62	62	61	62	61
G10 - Dev. Process & Governance (%)	60	60	60	60	60
G11 - Developers Base (%)	54	54	51	53	53
G12 - User Base (%)	73	72	74	73	70

On Figure 4.8, we can see that all four dimensions' score above 60% for all consecutive major releases. Therefore, following the resilience determination mechanism of our

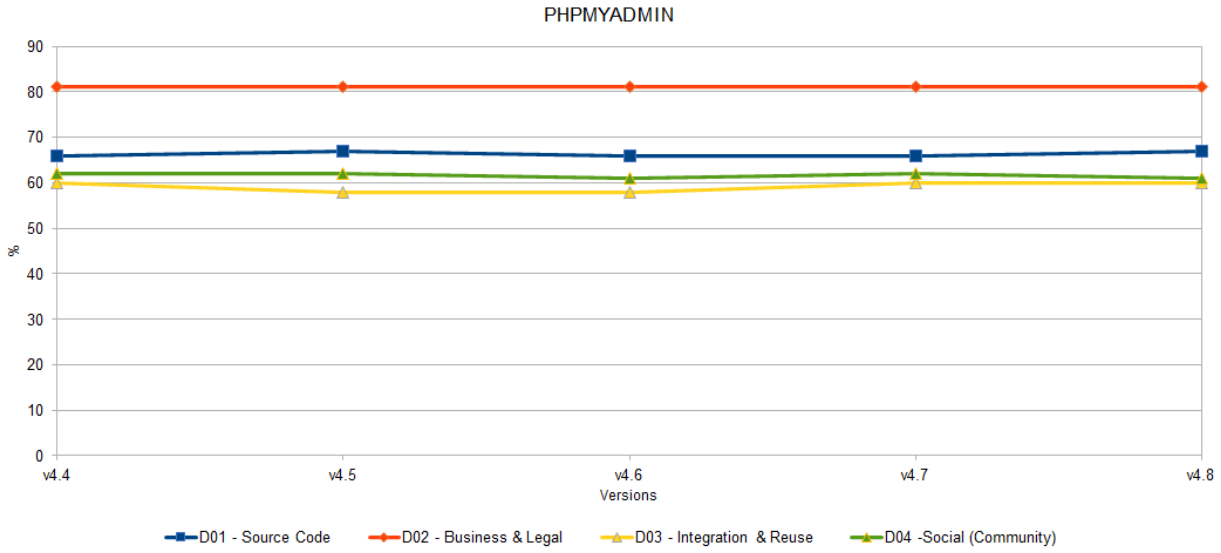


Figure 4.8: Resilience evolution between releases for PHPMyAdmin in OSSRF dimensions level

framework the project is considered resilient.

4.7.2 Non resilient projects

In the section we describe the application of the OSSRF to three (3) projects we intuitively classify as non resilient.

4.7.2.1 Okapi

It is a small framework for building web applications. It's written in PHP and is hosted in Github [20]. It started during 2008 and hasn't been updated since July 2011. To our best knowledge there is no license indication.

Since Okapi has no issue repository Effectiveness indicator it will receive a value of 0. Also, as we already mentioned all the qualitative indicators for this project are proposed with the average value of 2.

The rest of the indicators are measured using the tools described in the Section 6.2. The results of the application of OSSRF to Okapi are presented on TABLE 9.

In Figure 4.9, we can see that all four dimensions' score below 50% for all consecutive major releases. Moreover the trend of the four dimensions is also decreasing. Therefore, following the resilience determination mechanism of our framework the project is considered non resilient.

Table 4.6: OSSRF assessment for the OKApi project

	v1.1	v1.2
D01 - Source Code (%)	48	48
G01 - Architecture (%)	43	43
G02 - Maintainability (%)	40	40
G03 - Security & Testing (%)	61	62
D02 - Business & Legal (%)	27	27
G04 - License (%)	80	80
G05 - Market (%)	0	0
G06 - Support (%)	0	0
D03 - Integration (%) & Reuse	47	45
G07 - Initialization (%)	40	40
G08 - Dependencies (%)	40	40
G09 - Reuse (%)	60	54
D04 -Social (Community) (%)	25	15
G10 - Dev. Process (%) & Governance	20	20
G11 - Developers Base (%)	54	24
G12 - User Base (%)	0	0

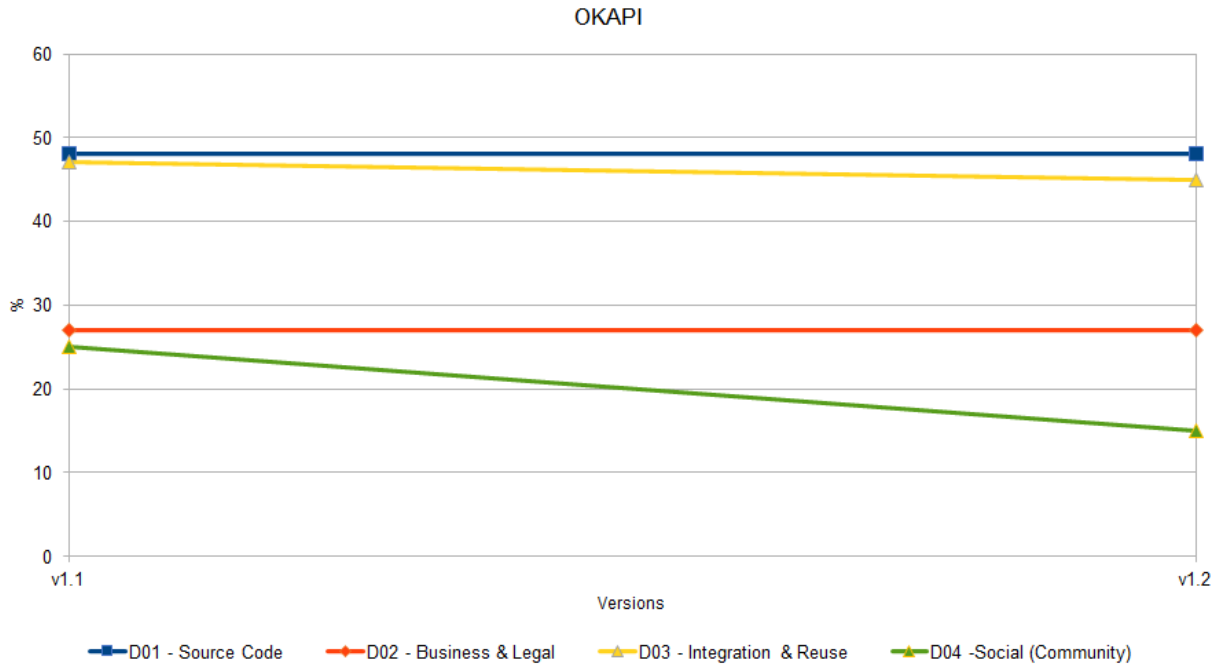


Figure 4.9: Resilience evolution between releases for OKApi in OSSRF dimensions level

4.7.2.2 PatternalPHP

PatternalPHP is written in PHP and is hosted in Github. It a lifetime less than two years however there are 5 conscutive releases that we studied. As we already mentioned all the qualitative indicators for this project are proposed with the average value of 2.

The rest of the indicators are measured using the tools described in the Section 6.2. The results of the application of OSSRF to PatternalPHP are presented on the following table.

Table 4.7: OSSRF assessment for the PatternalPHP project

	v0.3	v0.6	v0.7	v1.0	v2.0
D01 - Source Code (%)	33	34	34	33	33
G01 - Architecture (%)	43	43	43	43	43
G02 - Maintainability (%)	40	40	40	40	40
G03 - Security & Testing (%)	17	19	19	16	16
D02 - Business & Legal (%)	11	44	44	44	44
G04 - License (%)	0	100	100	100	100
G05 - Market (%)	0	0	0	0	0
G06 - Support (%)	33	33	33	33	33
D03 - Integration (%) & Reuse	47	47	47	45	42
G07 - Initialization (%)	40	40	40	40	40
G08 - Dependencies (%)	40	40	40	40	40
G09 - Reuse (%)	61	60	60	54	47
D04 -Social (Community) (%)	38	37	37	33	22
G10 - Dev. Process & Governance (%)	0	0	0	0	0
G11 - Developers Base (%)	61	66	64	60	38
G12 - User Base (%)	54	45	47	39	29

In Figure 4.10, we can see that all four dimensions' score below 50% for all consecutive major releases. Therefore, following the resilience determination mechanism of our framework the project is considered non resilient.

4.7.2.3 PHPExcel

PHPExcel is a PHP library that helps create and manage Excel files via PHP. Although currently an archived Github project, this specific software had attracted 65 contributors around it and has been forked more than 4,000 times on Github [22].

As we already mentioned all the qualitative indicators for this project are proposed with the average value of 2.

The rest of the indicators are measured using the tools described in the Section 6.2. The results of the application of OSSRF to PHPExcel are presented on the following table.

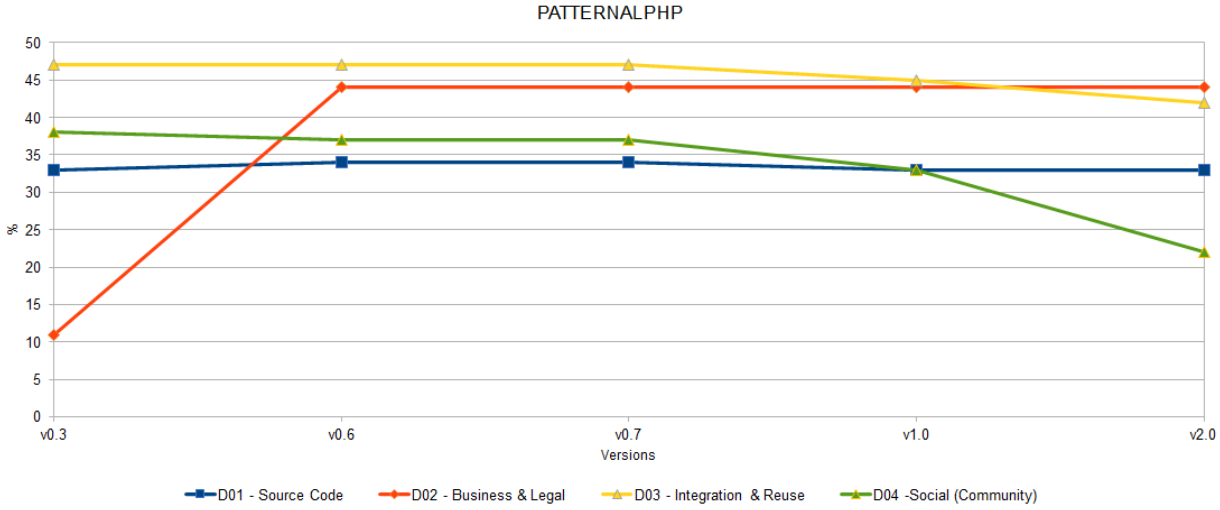


Figure 4.10: Resilience evolution between releases for PatternalPHP in OSSRF dimensions level

Table 4.8: OSSRF assessment for the PHPExcel project

	v1.7.8	v1.7.9	v1.8.0	v1.8.1	v1.8.2
D01 - Source Code (%)	50	50	49	48	47
G01 - Architecture (%)	43	43	43	43	43
G02 - Maintainability (%)	40	40	40	40	40
G03 - Security & Testing (%)	68	68	65	62	59
D02 - Business (%) & Legal	35	27	27	27	27
G04 - License (%)	80	80	80	80	80
G05 - Market (%)	25	0	0	0	0
G06 - Support (%)	0	0	0	0	0
D03 - Integration (%) & Reuse	42	42	41	42	39
G07 - Initialization (%)	40	40	40	40	40
G08 - Dependencies (%)	40	40	40	40	40
G09 - Reuse (%)	45	45	44	45	38
D04 -Social (Community) (%)	29	23	22	19	13
G10 - Dev. Process & Governance (%)	0	0	0	0	0
G11 - Developers Base (%)	50	56	53	36	38
G12 - User Base (%)	39	12	13	20	0

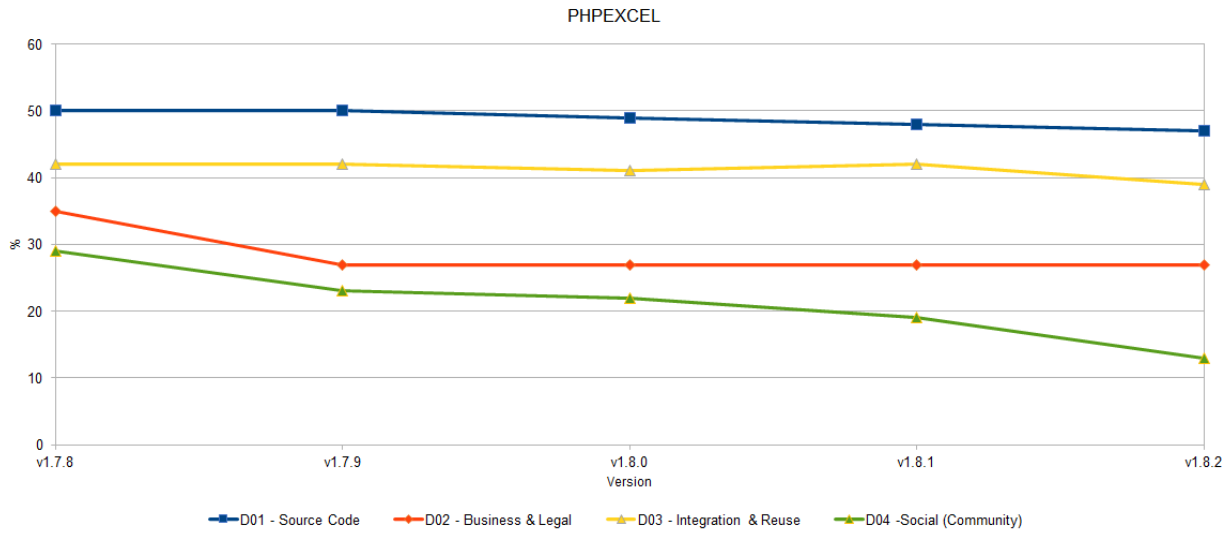


Figure 4.11: Resilience evolution between releases for PHPExcel in OSSRF dimensions level

In Figure 4.11, we can see that all four dimensions' score below 50% and present a declining trend. Therefore, following the resilience determination mechanism of our framework the project is considered non resilient.

NOTE: This part of our research was published by Wiley on the International Journal of Software: Evolution and Processes with the title: "A resilience-based framework for assessing the evolution of open source software projects". https://doi.org/10.1002/smr.2597

5. METRICS AGGREGATION TOOL (SOURCE-O-GRAPHER)

5.1 Software description

In this section we provide an analysis of the Source-o-grapher tool architecture, configuration and usage.

5.1.1 Software Architecture

Source-o-grapher is designed to run, as of the time of writing, on Linux operating systems. Most specifically the illustrative examples involved in this work ran in a machine with Ubuntu 22.04.1 LTS operating system. To install, configure and execute the software we use Linux shell scripts. A set of detailed steps is provided in the readme.md file that can be found in the repository of the software[1]. The collection of the input metrics (indicators) is provided by the user or handled semi-automatically by python scripts. The execution of the framework and output visualization is performed semi-automatically by python scripts.

The system was designed to be easily integrated with other systems. The current version of the tool was successfully integrated with the Github code repository and the PhpMetrics library.

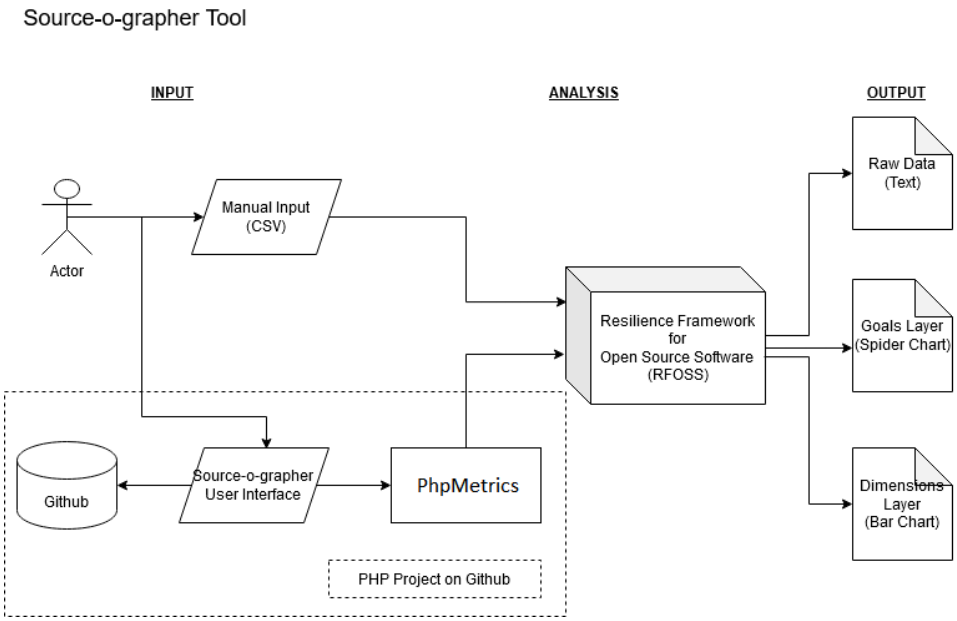


Figure 5.1: Source-o-grapher architecture

In Figure 5.1 you can see an abstract representation of the architecture of the tool. On

[1]Source-o-grapher Github repository: `https://github.com/AristotleUniversity/sourceographer`

the left side of the figure we can see the input process. There are two ways for the user to provide input to the tool:

1. Via a coma separated values (.CSV) formatted file. In this case the file contains a list of all the indicators that the tool uses (see Figure 5.2), and the user inputs the data to the file manually (this method does not utilize the automatic extraction of data from Github or PhpMetrics). This method is programming language agnostic. If the user utilizes this manual input process all they have to do is provide the values under the score column following the guidelines of the Open Source Software Resilience Framework[2]. The ”Indicator Name” and the ”Indicator ID” columns are used from the Source-o-grapher tool in order to assess the resilience of the OSS project when the manual input mode is used.
2. In cases where the OSS PHP project to be analyzed is hosted in Github, the tool provides a GUI for automatic analysis. Source-o-grapher is integrated with Github to automatically collect some of the indicators directly from the repository. It then processes another set of indicators using the PhpMetrics analyzer. The remaining indicators are being requested from the user via the GUI. Please note that the indicators requested from the user are qualitative indicators and therefore, since they represent expert opinion, cannot be automatically calculated. An example of this input is provided in Figure 5.3.

Then the indicators provided as input are being processed by the Resilience Framework for Open Source Software. This part of the calculation is handled automatically by the tool.

After the input is processed Source-o-grapher produces the following output:

- The results of the Resilience Framework for OSS alongside the input provided by the user. These are all included in the data.csv file that is also used for input reasons.
- The results for the goals are visualized as a spider chart and saved in PNG file format. An example of this output is presented in Figure 5.4.
- The results for the dimensions visualized as a bar chart. For the time being, this output is in the form of a PNG file. An example of this option of input is provided in Figure 5.5.

5.1.2 Integration with OSS Code Repositories

An integral part of Source-o-grapher is its integration with the Github code repository. It allows the tool to automatically acquire twenty (20) from the thirty eight (38) total indicators needed for the Software Resilient framework analysis.

[2]Open Source Software Resilience Framework - Indicators Documentation: `http://users.auth.gr/akritiko/ossrf`

	A	B	C
1	Indicator Name	Indicator ID	Score
2	Robustness	I01	0.6
3	Scalability	I02	0.6
4	Usability	I03	0.6
5	Effectiveness	I04	0.5
6	Corrections	I05	0.6
7	Improvements	I06	0.6
8	Security	I07	0.6
9	Testing process	I08	1
10	Coverage	I09	0.85
11	License type	I10	1
12	Dual licensing	I11	0
13	Commercial resources	I12	1
14	Commercial training	I13	1
15	Industry adoption	I14	1
16	Non profit / Foundation support	I15	1
17	For profit company support	I16	0
18	Donations	I17	1
19	Installability	I18	0.6
20	Configurability	I19	0.6
21	Self-contained	I20	0.6
22	Resource Utilization	I21	0.6
23	Complexity	I22	1
24	Modularity	I23	0.6
25	Instability	I24	0.45
26	Cohesion	I25	0.67
27	Governance model	I26	0
28	Project Road-map	I27	1
29	Code of conduct	I28	1
30	Coding standards	I29	1
31	Documentation standards	I30	1
32	Developers Attracted	I31	0.28
33	Active Developers	I32	0.27
34	Number of open issues	I33	0.95
35	Open / Closed issues	I34	0.91
36	Source Code Documentation	I35	0.43
37	Localization process	I36	1
38	Issue tracking activity (reporting bugs)	I37	0.05
39	User guide (completeness)	I38	1
40			

Figure 5.2: Input via a Comma Separated Values (CSV) formatted file.

Most of these indicators can be acquired directly by the API of Github. The remaining indicators must be calculated with the help of software analysis tools such as the PhpMetrics analyzer which we currently use for analyzing PHP based OSS projects. To be able to extract these information automatically from the Github repository of the project under investigation, the system needs the following input from the user:

- **Repository name in author/repo format:** The name of the project repository. In order to be able to separate two identically named repositories we require the repository in author/repo format (i.e. akritiko/sourceographer).
- **Version (repo specific tag name):** The name of the Github tag we want to analyze. This can help us investigate the resilience of the OSS project through its major releases as it evolves over time.
- **Repository URL:** The unified resource locator (URL) of the repository of the project

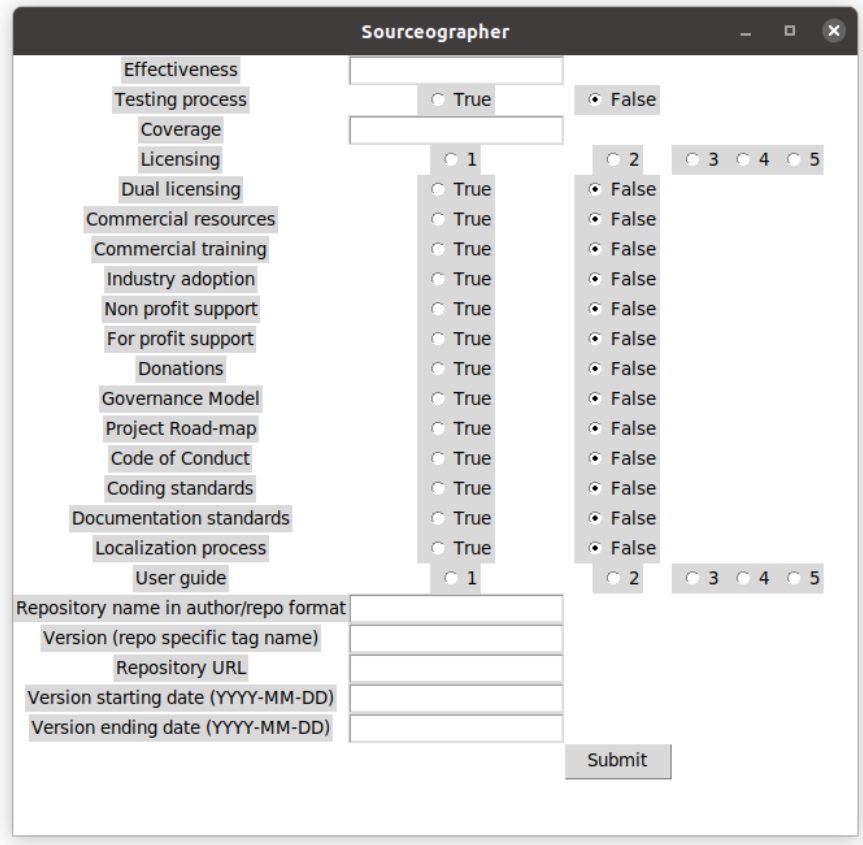


Figure 5.3: Input via the Graphic User Interface (GUI) of the tool.

under investigation.

- **Version starting date (YYY-MM-DD) & Version ending date (YYYY-MM-DD):** The starting and ending date of the version is used to be able to collect the time related indicators. For example the ”Developers Attracted” indicator on the Social dimension is calculated as the rate of developers that joined the project the last six months to the total numbers of the developer community of the project.

Apart from the ease of collecting the indicator values the integrated version of Source-o-grapher with the Github code repository has one more useful feature. It can automatically execute the Resilience Framework for OSS analysis to the Github version of our choice.

5.1.3 Integration with OSS analysis tools

Another core part of Source-o-grapher is its integration with OSS analysis tools. In the current version of the system we have integrated the PHPMetrics analyzer, a library that provides software analysis for projects implemented in the PHP programming language. This integration allows Source-o-grapher to automatically derive the values of some of the

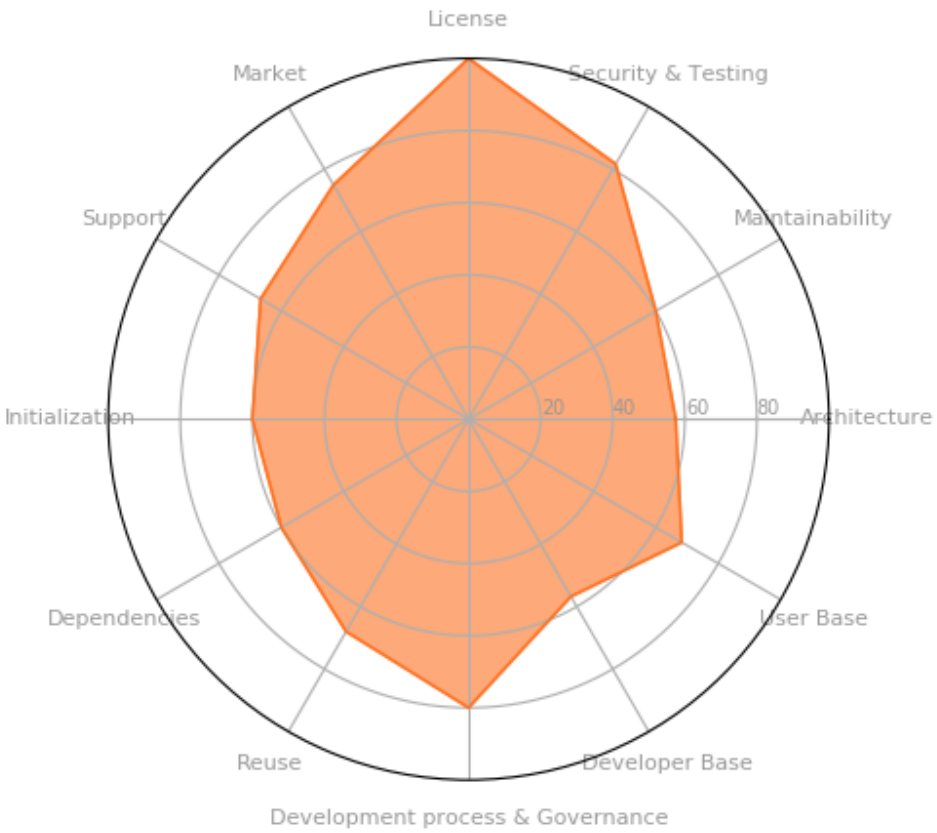


Figure 5.4: Output of goal scores as a radar chart.

indicators so that the user doesn't have to manually execute the analyzer and afterwards provide the results as an input to Source-o-grapher.

5.2 Illustrative Examples

In this section we are providing two examples of OSS projects assessed regarding their resilience with the use of Source-o-grapher. Composer (Illustrative Example #1) is a well known PHP Framework. We are using the manual input method for this project. PHPWord (Illustrative Example #2) is a small PHP library for creating and handling Word documents via PHP. We are using the Graphical User Interface input method of Source-o-grapher to provide the input for the Resilience Framework for OSS analysis.

We have chosed two diffrenet projects to showcase the two different types of input of source-o-grapher for better clarity. Also we aiming to provide the reader with a visual aid on how projects with different resilience assessments are presented on the the outputs of source-o-grapher (i.e. the radar charts of the two Illustrative Examples are indicative of

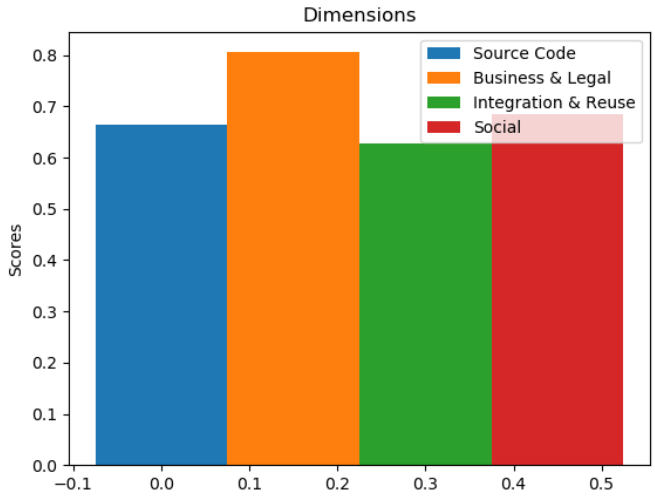


Figure 5.5: Output. Dimensions scores as a bar chart.

how Composer shows better resilience that PHPWord covering more of the radar charts surface).

Both the input and output data of the two illustrative examples are available in this Github repository[3].

5.2.1 Example #1: Manual input - Composer.

For this case study we are going to perform a Software Resilience analysis on the project Composer. Since we are using the manual way of importing data to Source-o-grapher we need to do the following:

1. In the software folder we navigate to the directory: ./rfoss/data.
2. There we can find the .csv file of the project (for our example we call that `composer_140.csv`; `140` reflects that this specific analysis is for version 1.4.0).
3. We open it with the editor of our choice and provide a value for each one of the indicators. In the specific version of the software you will find that this .csv file is pre-filled for your convinience. If you want to run Open Source Software Resilience Framework [60] (manually) for another OSS project, all you have to do is provide another set of inputs by cloning this .csv file an provide the metrics for the new software you are assessing. There is a detailed documentation[4] for the indicators values for the reader's convenience.

[3]Illustrative examples input and output: `https://github.com/AristotleUniversity/kritikos_softwarex_2022_examples`

[4]Open Source Software Resilience Framework - Indicators Documentation: `http://users.auth.gr/akritiko/ossrf`

4. After we are done, we save the file.
5. We navigate one directory up to the `./rfoss/` folder.
6. We open a console to this directory and we run the rfoss.py script using the command: `python3 rfoss.py`.
7. We navigate to the directory ./rfoss/output to find the output.

5.2.2 Example #2: Input using GUI - PHPWord

For this case study we are going to perform a Software Resilience analysis on the project PHPWord. Since we are using the semi-automated way of importing data to Source-ographer we need to do the following:

1. Environment Setup: Before we proceed to the analysis we need to make sure we have our environment configured correctly. To ensure that we nee to follow the instruction under the section ”Requirements & Installation” of the `README.md` file.
2. we navigate to `./src/data/` folder. We prepare the `data.csv` file per the instructions of the `README.md` file.
3. Once the file is ready, we navigate back to the `./src/` folder.
4. We open a console to this directory and we run: `python3 ossrf.py`.
5. When we run the command the GUI of the application will appear asking us for the input that cannot be automatically calculated. It is very important to fill all the information in this screen and use a valid Github Token, to avoid errors.
6. Once the process is completed, we can navigate back to the `./src/data/` folder to find our output. NOTE: Depending on the size of the OSS project we assess the script can take much time to finish. Please use the console log to make sure that the application is done.

The output consists of the following:

- The file `data.csv` containing the initial input of the user, the metrics automatically retrieved by Github and PhpMetrics and the Goals and Dimensions results from the Open Source Software Resilience Framework.
- A bar plot visualizing the results of the framwork to the Dimensions Level.
- A radar chart visualizing the results of the framework to the Goals’ level.
- A `.json` file containing the PhpMetrics indicators for the specific project under assessment. This file can be used by researcher for further cross-check analysis with other assessment or evaluation tools.

5.2.3 Software Resilience Assessment

Now we can assess the results of the two case studies to get an idea of what the Software Resilience Analysis can tell us for the two OSS projects. In order to be able to better compare the two projects we will be studying the results on dimension and goal level together.

5.2.3.1 Dimension results

Figures 5.8 and 5.9 present the dimension results for Composer and PHPWord project respectively. Each of the bars in the bar chart represent a dimension and the height of the bar is presenting the percentage of the fulfillment in percentage. By comparing the two charts we conclude:

- Both projects are considered resilience in terms of dimensions. Composer scores around 60% or higher for all dimensions. PHPWord score low (below 30%) for two dimensions but the other two score more than 60%. Therefore, since two out of four dimensions score more than 50% the project is considered resilience.
- We can see that resilience is different to the different dimensions of the two projects. Composer, one of the most known PHP projects scores much higher than PHPWord in the dimensions of Bussiness & Logic and Social. Composer is a very mature project with a vibrant community and, at the same time it is being used by several corporations for faster development. Therefore, its excellence over PHPWord is intuitively expected.

5.2.3.2 Goals results

The difference in terms of the resilience of the two projects becomes more evident if we compare the results on goals level. In Figures 5.10 and 5.11 we can see the results for the projects Composer and PHPWord respectively.

The radar charts present the twelve goals. The more the highlighted territory, the more resilient the project in each goal.

A1 | fx Σ = Indicator Name

	A	B	C
1	Indicator Name	Indicator ID	Score
2	Robustness	I01	0.6
3	Scalability	I02	0.6
4	Usability	I03	0.6
5	Effectiveness	I04	0.5
6	Corrections	I05	0.6
7	Improvements	I06	0.6
8	Security	I07	0.6
9	Testing process	I08	1
10	Coverage	I09	0.89
11	License type	I10	1
12	Dual licensing	I11	0
13	Commercial resources	I12	1
14	Commercial training	I13	1
15	Industry adoption	I14	1
16	Non profit / Foundation support	I15	1
17	For profit company support	I16	1
18	Donations	I17	0
19	Installability	I18	0.6
20	Configurability	I19	0.6
21	Self-contained	I20	0.6
22	Resource Utilization	I21	0.6
23	Complexity	I22	0.8
24	Modularity	I23	0.6
25	Instability	I24	0.54
26	Cohesion	I25	0.67
27	Governance model	I26	0
28	Project Road-map	I27	1
29	Code of conduct	I28	1
30	Coding standards	I29	1
31	Documentation standards	I30	1
32	Developers Attracted	I31	0.11
33	Active Developers	I32	0.33
34	Number of open issues	I33	0.95
35	Open / Closed issues	I34	0.96
36	Source Code Documentation	I35	0.8
37	Localization process	I36	1
38	Issue tracking activity (reporting bugs)	I37	0.05
39	User guide (completeness)	I38	1
40			

Figure 5.6: Manual input based on the .csv template. Composer v1.4.0.

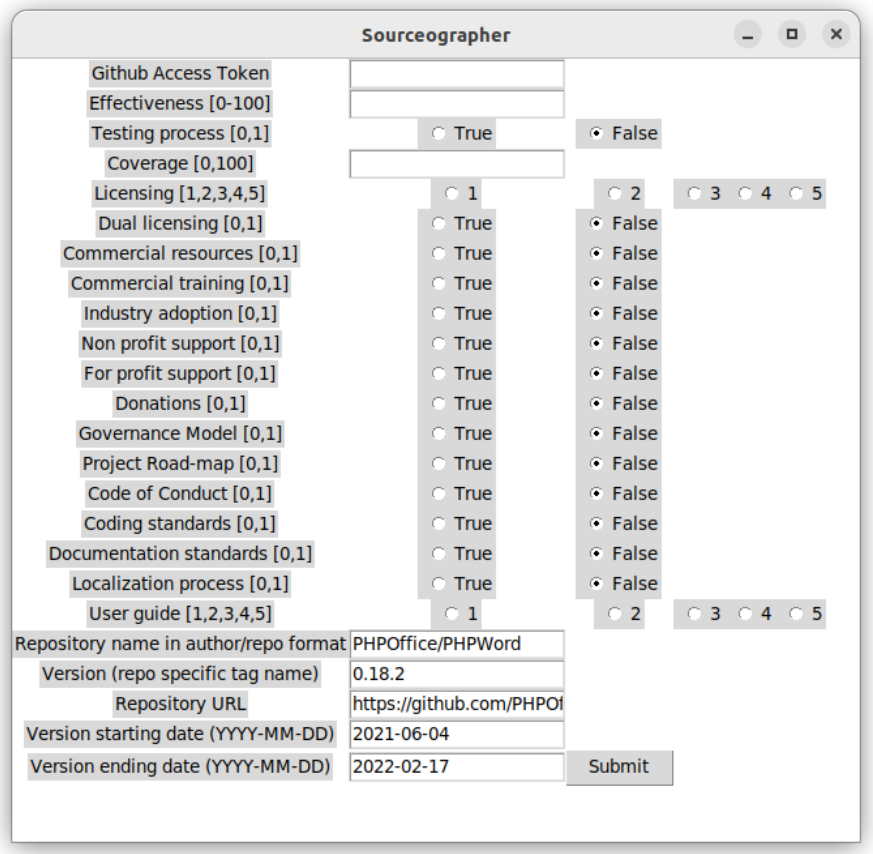


Figure 5.7: Graphical User Interface. Indicators and project properties.

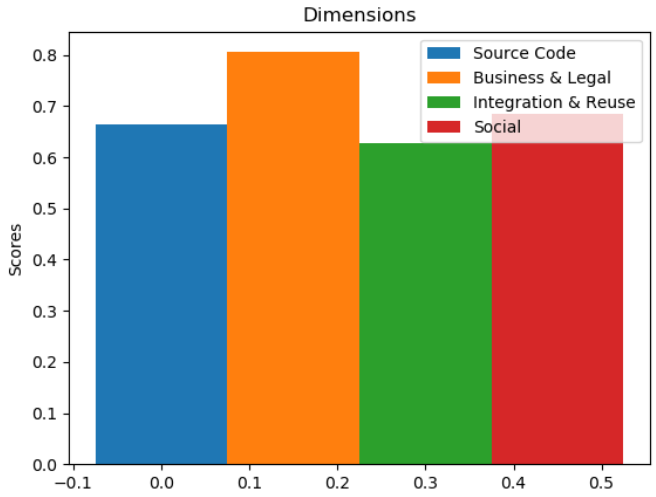


Figure 5.8: Composer. Dimensions results

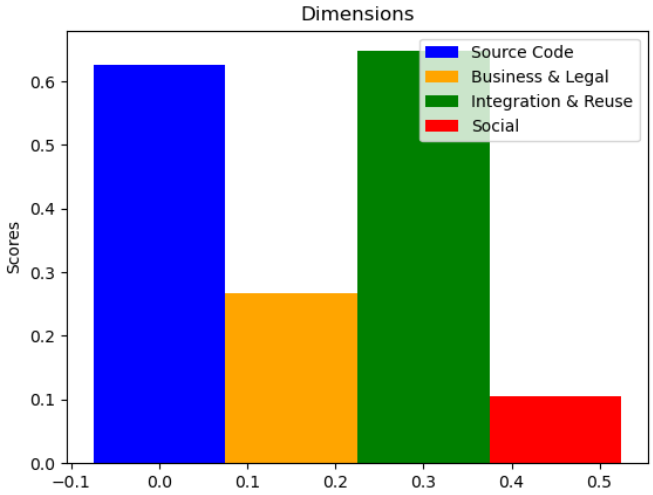


Figure 5.9: PHPWord. Dimensions results

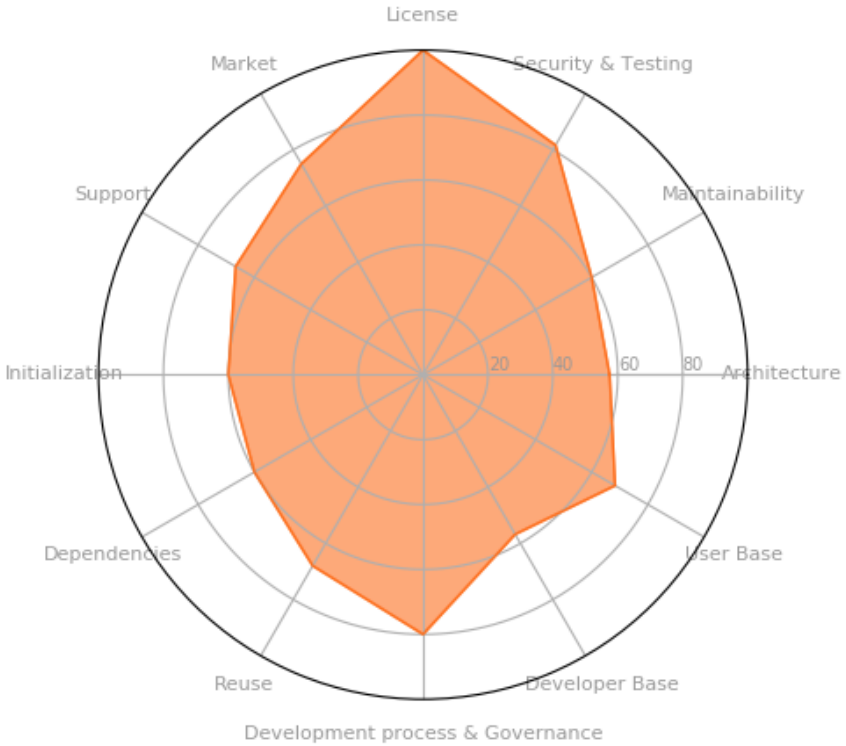


Figure 5.10: Composer. Goals results

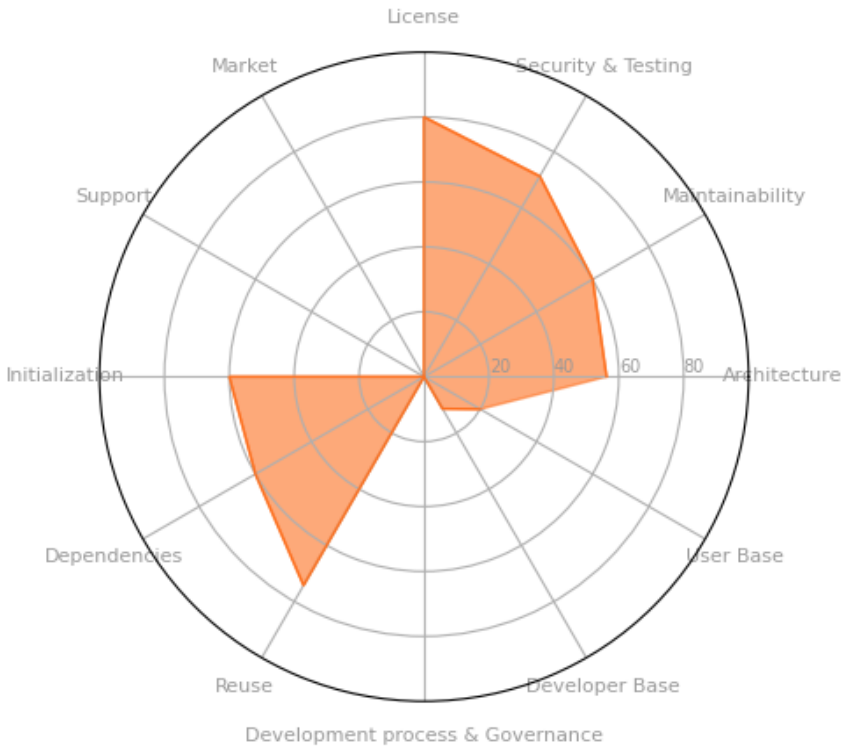


Figure 5.11: PHPWord. Goals results

The results merely validate the findings from the dimensions cross-check analysis. We can clearly see that the radar chart of the Composer project scores better in most of the goals something that is evident from the coverage of the radar chart compared to the coverage of the radar chart of PHPWord.

NOTE: This part of our research was published by Elsevier on the International Journal SoftwareX with the title: "Source-o-grapher: A tool towards the investigation of software resilience in Open Source Software projects". `https://doi.org/10.1016/j.softx.2023.101337`

6. EPISTEMIC ANALYSIS

The scientific framework of the analysis conducted is the scientific framework of software engineering. As we discussed earlier, a scientific framework consists of five fields: Epistemology, Identity, Knowledge, Skills, and Values. For each field of the scientific framework, certain codes are defined. These codes were determined using the Software Engineering Body of Knowledge (SWEBOK) [35], which describes generally accepted knowledge about software engineering and is freely available online. The codes we arrived at and used in the analysis are as follows:

Category	**Sub-Items**
Epistemology	consultant / software engineer, data, design, client / user, language
Identity	architect, associate / partner, developer, software engineer, contributor, junior / entry, senior, leader, project-manager / owner, tester
Knowledge	error-handling, software model, architecture, abstraction, business domain, technical, design specs, dev-environment, data, design, runtime-issues, design-patterns, language, variable types
Skills	abstraction, analyze, collaboration, communication-skills, cost-estimation, data, debug, design, develop, knowledge of project code and functionality, planning, problem-solving, professional, risk-assessment, test
Values	agility, client / user, quality, security, usability, innovation, reusability

Table 6.1: Epistemological Analysis field codes

6.1 Dialogue selection and coding and ENA WebKit analysis

As mentioned above, the open source projects selected for the analysis are OpenOffice and LibreOffice. More specifically, from these projects, we chose the following dialogues:

The first step of the analysis was the encoding of the dialogues. More specifically, the dialogue lines were first recorded in a .csv file, categorized by project and dialogue bug/number. Subsequently, all the codes of the scientific framework were added as columns. For each dialogue line, we set the value one (1) in the cells of the codes where there was considered to be conceptual correlation, and a zero was placed in those that did not have it, as shown in the indicative example in Figure 6.1.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	UserNam	Project	Bug	Timestamp	text	E.consulta	E.data	E.design	E.client / u	E.language	I.architect	I.associate	I.develope	I.software
2	sberg	LibreOffice	1_1	1/29/2020	In C/C++, if	0	0	0	0	1	0	0	0	1
3	sberg	LibreOffice	1_1	1/29/2020	That is why t	0	1	0	0	0	0	0	0	0
4	sberg	LibreOffice	1_1	1/29/2020	Which has th	0	1	0	0	0	0	0	0	0
5	sberg	LibreOffice	1_1	1/29/2020	Another drav	0	0	0	0	0	0	0	0	0
6	sberg	LibreOffice	1_1	1/29/2020	Enter	0	0	0	0	0	0	0	0	0
7	sberg	LibreOffice	1_1	1/29/2020	If e1 is	0	0	0	0	0	0	0	1	1
8	sberg	LibreOffice	1_1	1/29/2020	(o3tl::make	0	0	0	0	0	0	0	1	0
9	sberg	LibreOffice	1_1	1/29/2020	The caveat	0	0	0	0	0	0	0	1	0
10	sberg	LibreOffice	1_1	1/29/2020	<https://ger	0	0	0	0	0	0	0	0	0
11	sberg	LibreOffice	1_1	1/29/2020	There is a	0	0	0	0	0	0	0	0	0
12	sberg	LibreOffice	1_1	1/29/2020	<https://ger	0	0	0	0	0	0	0	1	0
13	sberg	LibreOffice	1_1	1/29/2020	But I may	0	0	0	0	0	0	0	1	0
14	sberg	LibreOffice	1_1	1/29/2020	So if you	0	0	0	0	0	0	0	0	0
15	Lubos Lun	LibreOffice	1_1	1/29/2020	On	0	0	0	0	0	0	0	0	0
16	Lubos Lun	LibreOffice	1_1	1/29/2020	I think the pr	0	0	0	0	0	0	0	1	0
17	Lubos Lun	LibreOffice	1_1	1/29/2020	Technically it	0	1	0	0	0	0	0	0	1
18	Lubos Lun	LibreOffice	1_1	1/29/2020	But IM(NSH)I	0	0	0	0	0	0	0	0	0
19	Lubos Lun	LibreOffice	1_1	1/29/2020	There are or	0	1	1	0	1	0	0	0	0
20	Lubos Lun	LibreOffice	1_1	1/29/2020	Trying to squ	0	0	0	0	0	0	0	0	0
21	Lubos Lun	LibreOffice	1_1	1/29/2020	And trying to	0	1	0	0	1	0	0	0	0
22	Lubos Lun	LibreOffice	1_1	1/29/2020	The real fix t	0	0	0	0	0	0	0	1	1
23	Lubos Lun	LibreOffice	1_1	1/29/2020	If the idea is	0	0	0	0	0	0	0	0	0
24	Lubos Lun	LibreOffice	1_1	1/29/2020	So while I like	0	0	0	0	0	0	0	0	1
25	sberg	LibreOffice	1_1	1/29/2020	On	0	1	0	0	0	0	0	0	0

Figure 6.1: Indicative example of the coding of dialogues

The following steps of ENA consist of the creation of graphs and consequently the visualization of networks, were performed using an online tool called ENA WebKit [12]. The ENA WebKit performs two main functions:

1. It processes encoded data:
    - Takes the data table
    - Divides the lines into stanzas
    - Accumulates codes per stanza
    - Generates a set of adjacency matrices
    - Creates an aggregated adjacency matrix representing the connections between encoded objects for each unit of analysis
    - Produces dimensionality reduction for data representation
2. It uses the results of this analysis to generate visualizations that facilitate data exploration and interpretation.

    Initially, the selection was made for:

    - Units: Units can refer to individuals, concepts, groups, or anything for which the ENA online tool wants to shape the network connections. In other words, units are the pieces of our data for which ENA constructs networks. Selected variable columns representing our units are shown in Figure 6.2. For each line in the

data, unit variables indicate to which unit the line belongs. By default, ENA constructs networks for each unique unit defined by the selected columns.

- Conversation: Conversations are collections of lines where ENA models connections between concepts. For example, we might want to model connections in different time segments, like days, or different steps in a process, such as activities, or in our case, different bugs. Concepts and units in lines not in the same conversation are not related to each other in the model. Selected conversation variable columns are shown in Figure 6.2. A conversation variable indicates, for each data line in each unit, to which conversation it belongs. By default, ENA models connections in each unique conversation defined by the selected columns.

- Stanza Window: There are two primary ways to model conversations in ENA: (1) using the entire conversation or (2) using a stanza window. Using the entire conversation means that all lines in a given conversation are related, and ENA will model connections in all these lines. Using a moving stanza window means that ENA will model connections within the conversation by dividing it into multiple stanzas, i.e., collections of lines within a conversation. Here, lines are considered to be related only within a defined stanza window. In other words, this way we model connections between lines that are in close temporal proximity within a conversation or in a close relationship in terms of another measurement unit. The chosen conversation method is shown in Figure 6.2.

- Codes: Codes are concepts whose association patterns we want to model. ENA represents connections between the chosen codes as networks for units and groups. Selected data columns containing values for each line are shown in Figure 6.3. The values are binary (0 or 1).

- Optional Comparison: Groups are collections of units whose networks we want to compare. For example, we might want to compare networks of different units by gender or by experimental condition. If we consider having groups in our data, we select a column containing grouping variables. For each data line, grouping variables indicate which group the line belongs to. Group selection is an optional step. By default, ENA will construct and plot networks for the two largest grouping variables from the selected column. The selected grouping variable in the first experiment is shown in Figure 6.2.

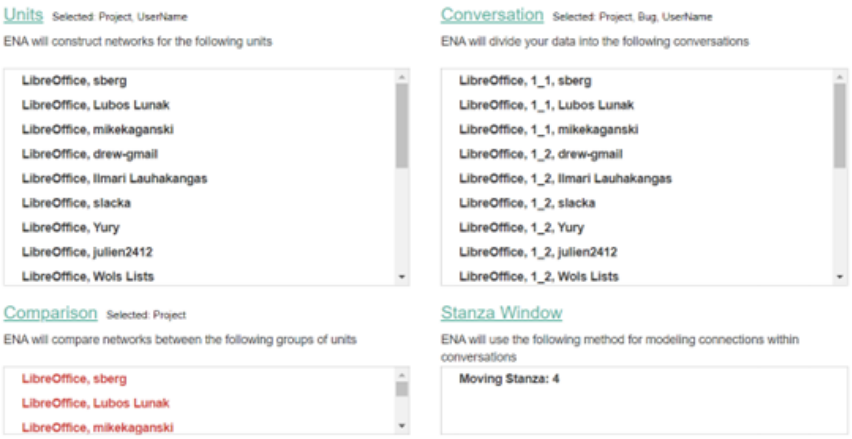


Figure 6.2: The configuration of ENA WebKit (units, conversation, comparison, stanza window) for the first experiment

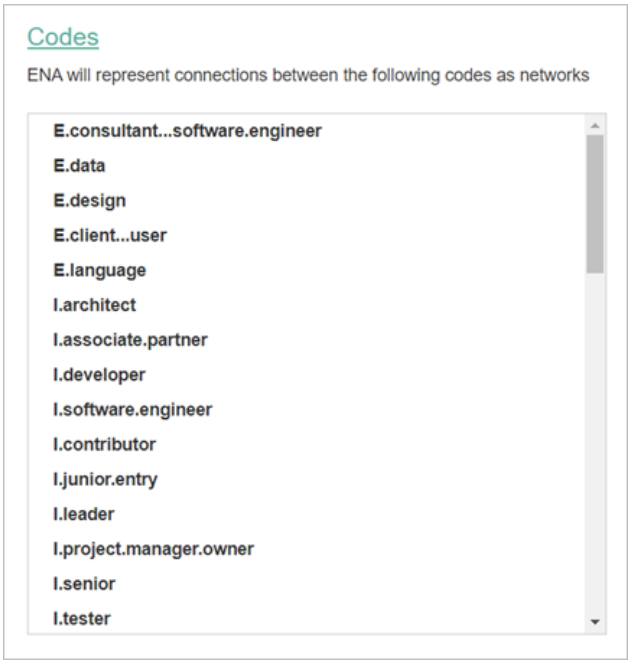


Figure 6.3: Code selection at ENA WebKit for all experiments

6.2 Applying Epistemic Network Analysis to Open Source Software Projects

In this section, we will present three (3) experiments conducted using the ENA WebKit and the encoded dialogs discussed above.

6.2.1 Experiment 1: Comparison of the two projects

The parameter selections for this specific experiment include the following parameter choices:

1. As units, the columns ”Project” and ”Username”.
2. As conversation, the columns ”Username,” ”Project,” and ”Bug.”
3. As stanza window, a moving window of four lines.
4. All codes from the .csv file were selected as codes.
5. As comparison, the ”Project” column.

The first image we see in the ENA WebKit, after selecting the parameters, is the one shown in Figure 6.4, where we can also observe the confidence intervals for the two projects:

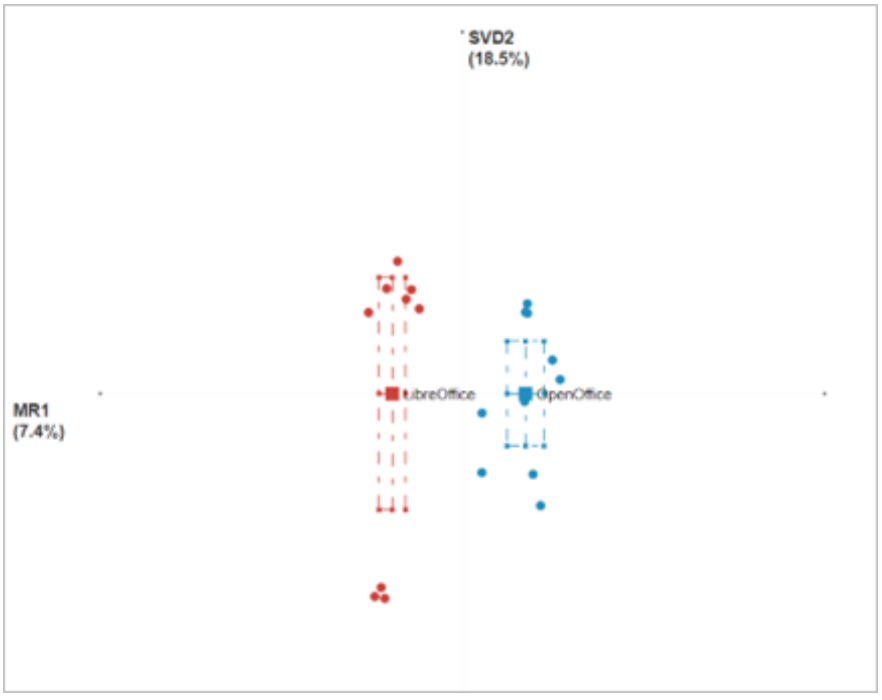


Figure 6.4: A first comparison of the two projects

By default, the ENA space in which the centroids are displayed is determined by the first (x) and second (y) dimensions, representing the dimensions with the greatest data variance. (Remember that the unique decomposition value maximizes variance during dimension reduction.) The numbers in parentheses next to the axis labels indicate the percentage of data variance for those dimensions. In this case, dimension x represents 7.4% of the data variance and dimension y represents 18.5%.

As shown in Figure 6.5, there is a significant difference between the two projects. To determine the difference more precisely, we can perform an independent samples t-test. To do this, we just need to select the two samples we want to compare from the dropdown menu on the left in the ”Stats” tab (see Figure 6.6). When we do this, we will see the means for the two samples, along with the t-score, p-value, and Cohen’s d, a measure of effect size. In this case, the difference in the first dimension is significant:

Along the X-axis, the two-sample t-test, assuming unequal variances, showed that OpenOffice (mean = 1.06, SD = 0.44, N = 10) was statistically significantly different at the alpha level of 0.05 from LibreOffice (mean = 1.18, SD = 0.29, N = 9; t(15.80) = 13.32, p = 0.00, Cohen’s d = 5.99).

Additionally, we can calculate the strength of the correlation between centroids and projected points in the model, using both Pearson’s r and Spearman’s r_s. In this case, both are equal to 1 for both dimensions, because the number of units in the model is small compared to the number of dimensions (see Table 6.3). Optimization is therefore easy to resolve.

	Pearson	**Spearman**
X Axis:	1.00	1.00
Y Axis:	1.00	1.00

Table 6.3: Pearson’s and Spearman’s Corellation Coefficients

To determine what this difference represents in the structures of connections, we can examine the equiload projections. Clicking on the centroid of a username allows us to view that network, but we can also click on a mean to see the average network for all usernames in that particular project. By doing so, we obtain the networks shown in Figures 6.5 and 6.6.

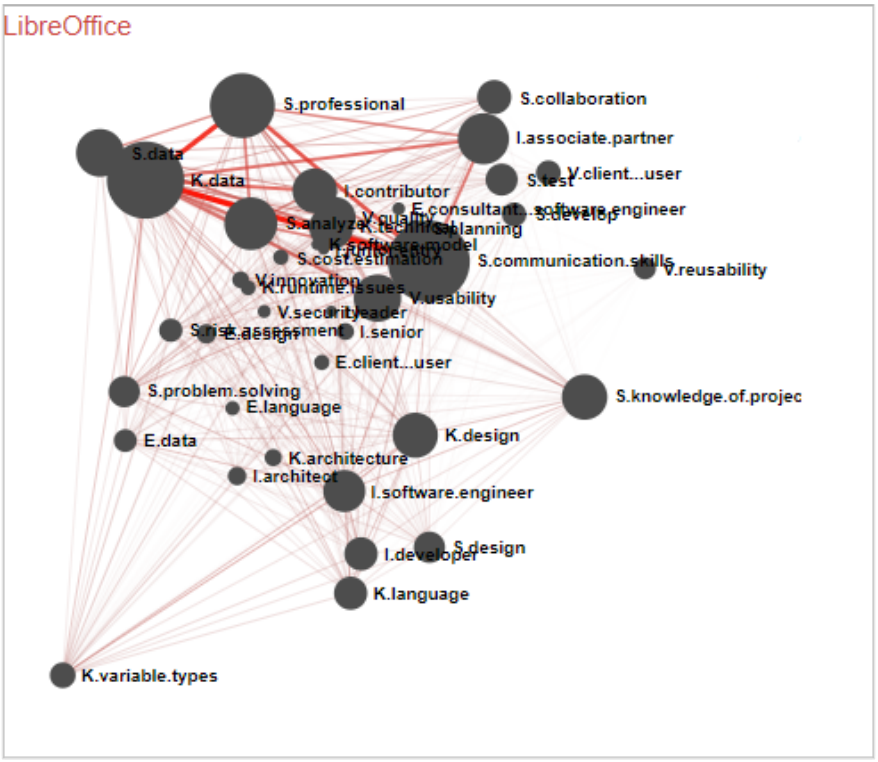


Figure 6.5: Average network for LibreOffice

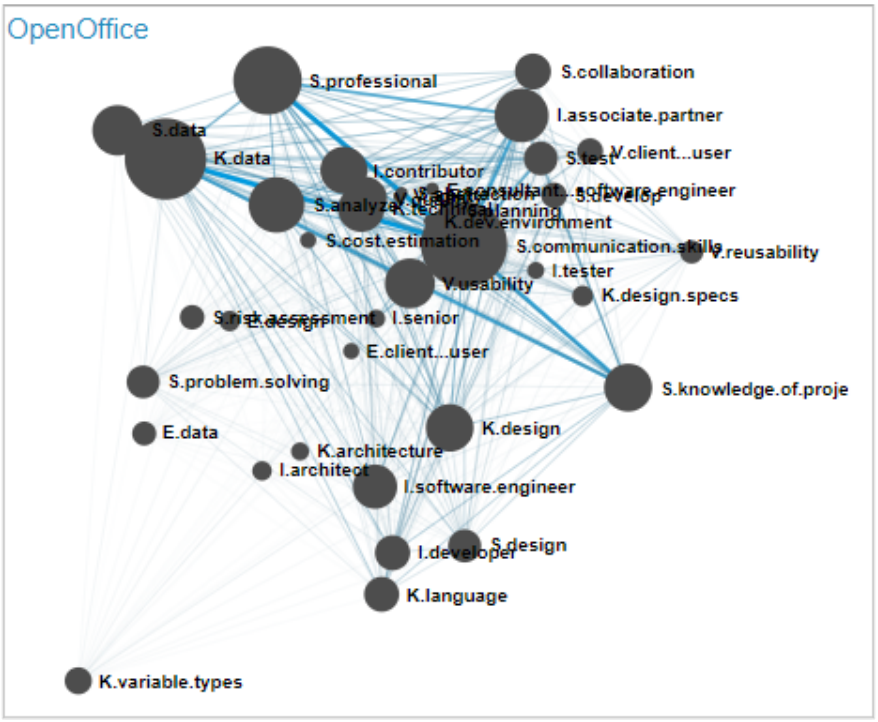


Figure 6.6: Average network for OpenOffice

Based on the network analyses of the two projects, in the context of the experiments presented, the most significant difference appears to be that participants in LibreOffice formed stronger connections between their knowledge of data and other interpersonal skills (such as “communication skills”), as well as other skills like professionalism and analytical ability. Additionally, they showed connections with the identities of “contributor” and “associate/partner,” and the value of usability. On the other hand, OpenOffice exhibited stronger connections between knowledge of data and the ability to understand code and project functionality, along with communication skills and the ability to understand technical aspects. Moreover, OpenOffice displayed robust connections between communication skills and professional behavior.

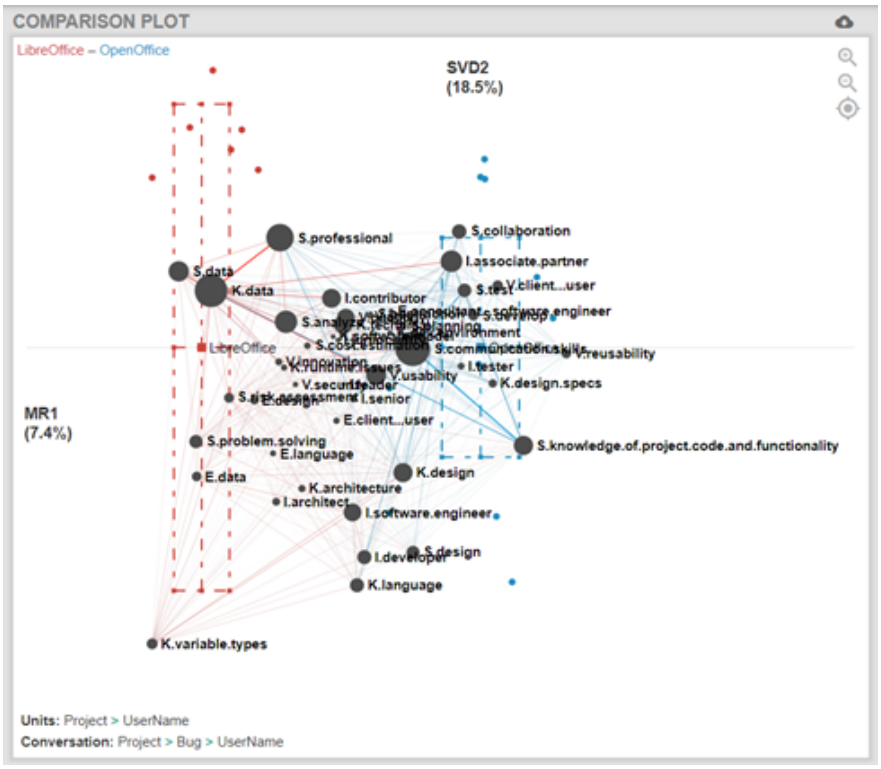


Figure 6.7: Comparison model for the LibreOffice and OpenOffice networks

In general, from Figure 6.7, it is evident that LibreOffice has stronger connections than OpenOffice, and its connections extend across more domains of the epistemic frame. Consequently, we could say that LibreOffice exhibited more scientific dialogues.

6.2.2 Experiment 2: Comparison of bugs for the two projects

The parameter selections for this specific experiment include the following selections:

- As units, the columns ”Bug” and ”Username”,

- As conversation, the columns ”Username,” ”Project,” and ”Bug”,

- As stanza window, a moving window of four lines,

- All codes from the .csv file were selected as codes,

- As comparison, the ”Bug” column.

Below, we can see the centroids and confidence intervals for the bugs of the 1st project (see Figure 6.8) and the bugs of the 2nd project (see Figure 6.9):

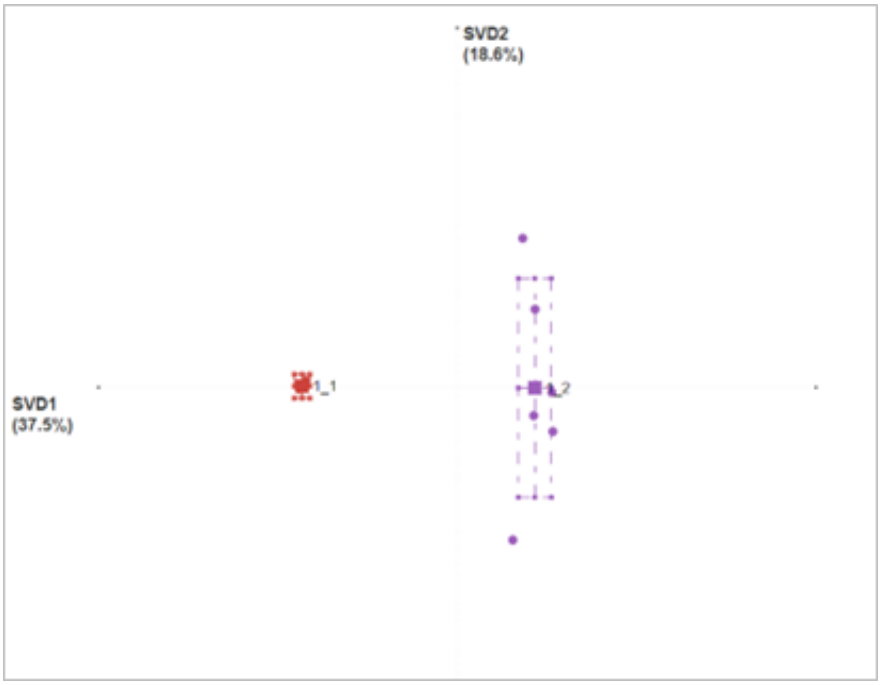


Figure 6.8: Centroids and confidence intervals for the bugs of LibreOffice.

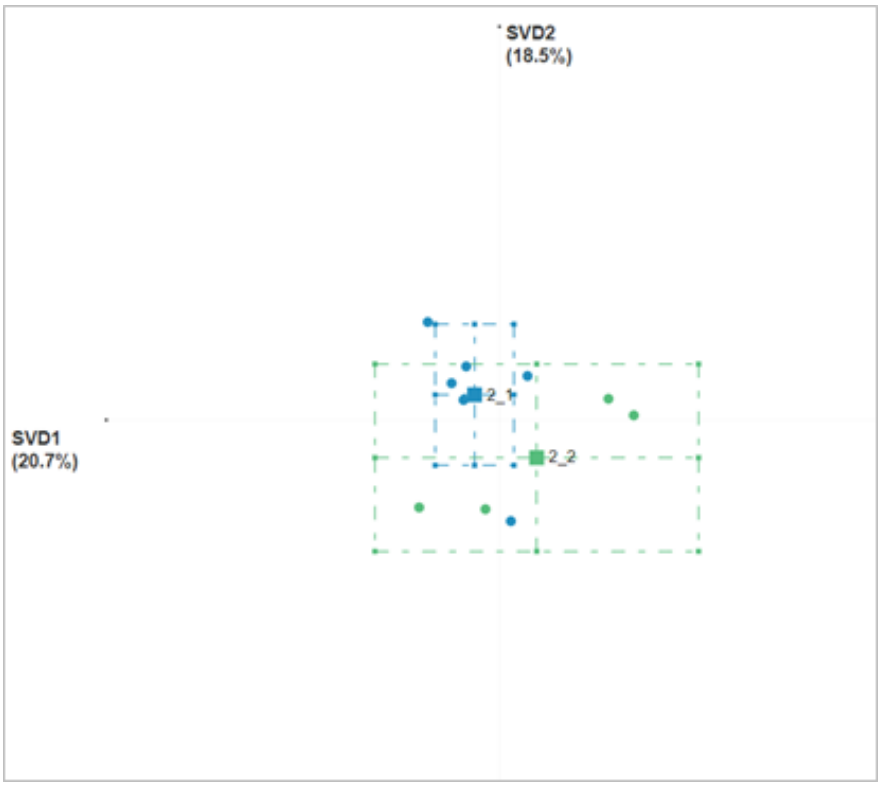


Figure 6.9: Centroids and confidence intervals for the bugs of OpenOffice.

Note that the dimensions are different from the previous example. We need to remember that ENA creates the space in which networks are visualized based on the parameters used to create the dataset. Since we are examining bugs and not projects, the space is different.

As shown in Figure 6.8, there is a significant difference between the two bugs of LibreOffice. To determine the difference more accurately, we conducted an independent samples t-test. By performing the test, we observed that the difference in the first dimension is significant.

Along the X-axis, the two-sample t-test, considering unequal variances, showed that bug 1_1 (mean = -2.21, SD = 0.04, $N = 3$) was statistically significantly different at the alpha level of 0.05 from bug 1_2 (mean = 1.10, SD = 0.22, $N = 6$; $t(5.73) = -34.78$, $p = 0.00$, Cohen's $d = 17.31$).

Just like in the previous experiment, both Pearson's and Spearman's r are equal to 1 for both dimensions because the number of units in the model is small compared to the number of dimensions (see Table 6.4).

To determine what this difference represents in the structures of connections, we can examine the network projections, as done previously (see Figures 6.10 and 6.11).

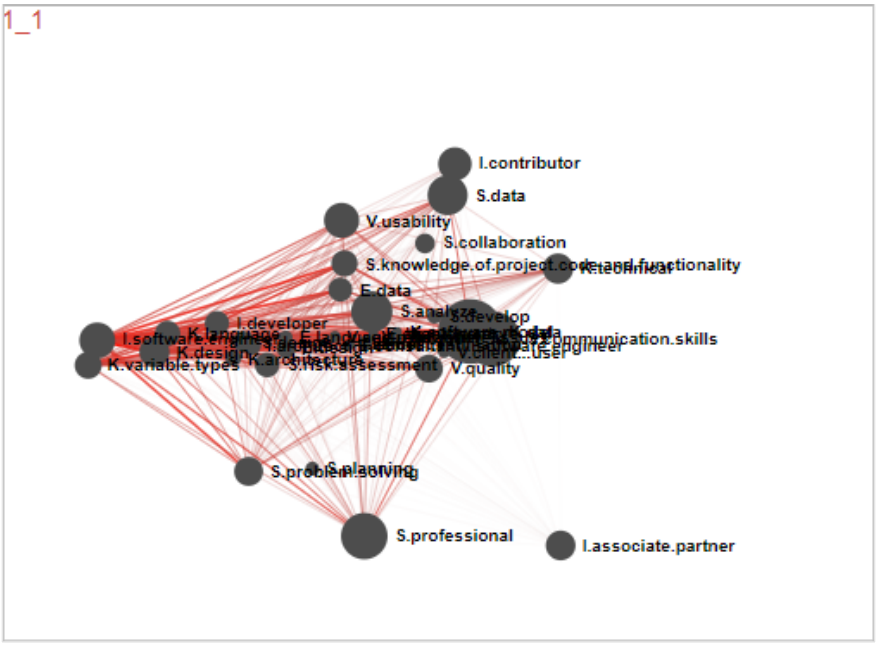


Figure 6.10: Average network for the first bug of LibreOffice.

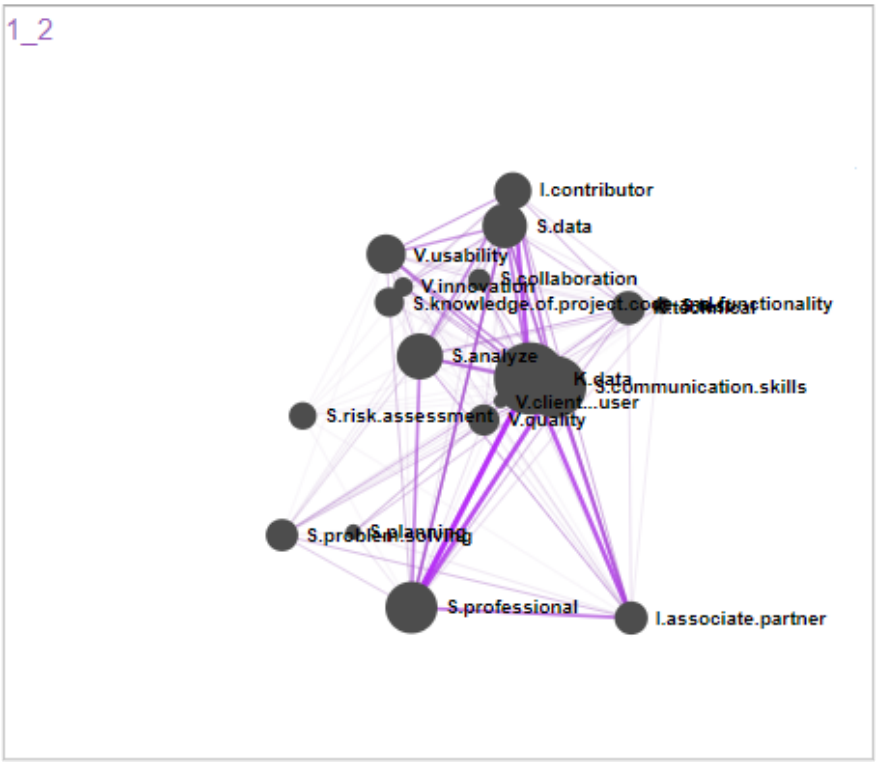


Figure 6.11: Average network for the second bug of LibreOffice.

Based on the average networks of the two bugs, the most significant difference seems to be that in bug 1_1, the participants created stronger connections between the identity of

”software engineer” and the knowledge of design, knowledge of variable types, knowledge of data, and communication skills. Strong connections were also observed between the identity of ”software engineer” and the identity of ”developer,” as well as with knowledge of the project’s code and functionality, and knowledge of language. The ability to understand code and project functionality had a strong connection with the knowledge of design, as did the latter with the knowledge of variable types. Additionally, an important observation is that in this bug, participants established a strong connection between the identity of ”software engineer” and the epistemology of data, indicating that their reasoning was based on the epistemology concerning the data they were discussing.

In contrast, in bug 1_2, stronger connections were observed between the identity of ”associate/partner” and communication skills, as well as knowledge of data. Furthermore, strong connections existed between the ability of professionalism and communication, as well as with knowledge of data and the ability to produce and present data. Very strong connections were also present between the identity of ”contributor” and knowledge of data, as well as communication skills, professionalism, the ability to produce and present data, the ability to analyze, and the value of usability.

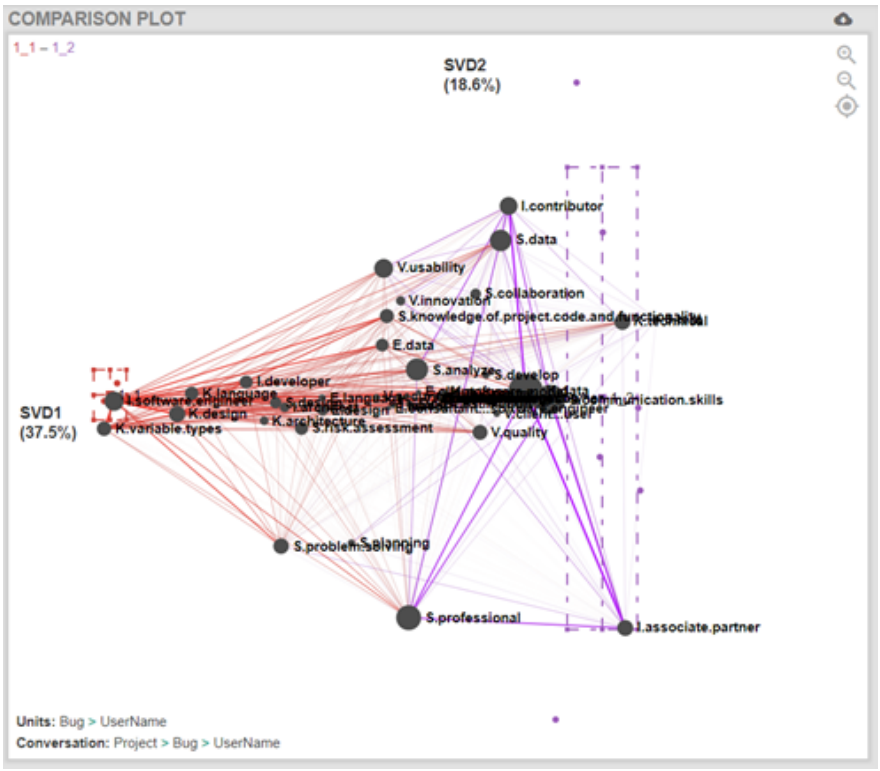


Figure 6.12: Comparison model for the two bugs of LibreOffice.

From Figure 6.12, we understand that the two networks extend to different sides of the epistemic frame, with the network of bug 1_1 focusing on fields related to knowledge and certain skills, as well as epistemology. On the other hand, the network of 1_2 connects more fields related to skills. Therefore, we could say that bug 1_1 presents more scientific

dialogues, and it would be beneficial for participants in bug 1_2 to use more epistemology in their discourse.

As we can also perceive from Figure 6.11, there doesn't seem to be a significant difference between the two bugs of OpenOffice. By performing an independent sample t-test, we observe that there is no statistically significant difference in any dimension:

Along the X-axis, the two-sample t-test, considering unequal variances, showed that bug 2_1 (mean = -0.37, SD = 0.56, N = 6) was not statistically significantly different at the alpha level of 0.05 from bug 2_2 (mean = 0.56, SD = 1.53, N = 4; t(3.55) = -1.17, p = 0.31, Cohen's d = 0.90).

Along the Y-axis, the two-sample t-test considering unequal variances showed that bug 2_1 (mean = 0.38, SD = 1.01, N = 6) was not statistically significantly different at the alpha = 0.05 level from bug 2_2 (mean = -0.57, SD = 0.88, N = 4; t(7.21) = 1.56, p = 0.16, Cohen's d = 0.98).

As seen in Figure 6.13, the network of bug 2_1 extends towards the domains of design knowledge, usability value, professionalism, and communication skills. Similarly, the network of bug 2_2 extends more towards the identity of a software engineer, language knowledge, the identities of "senior" and "associate / partner," and communication skills. However, the comparison conducted by the independent t-test indicated that their differences were not statistically significant.

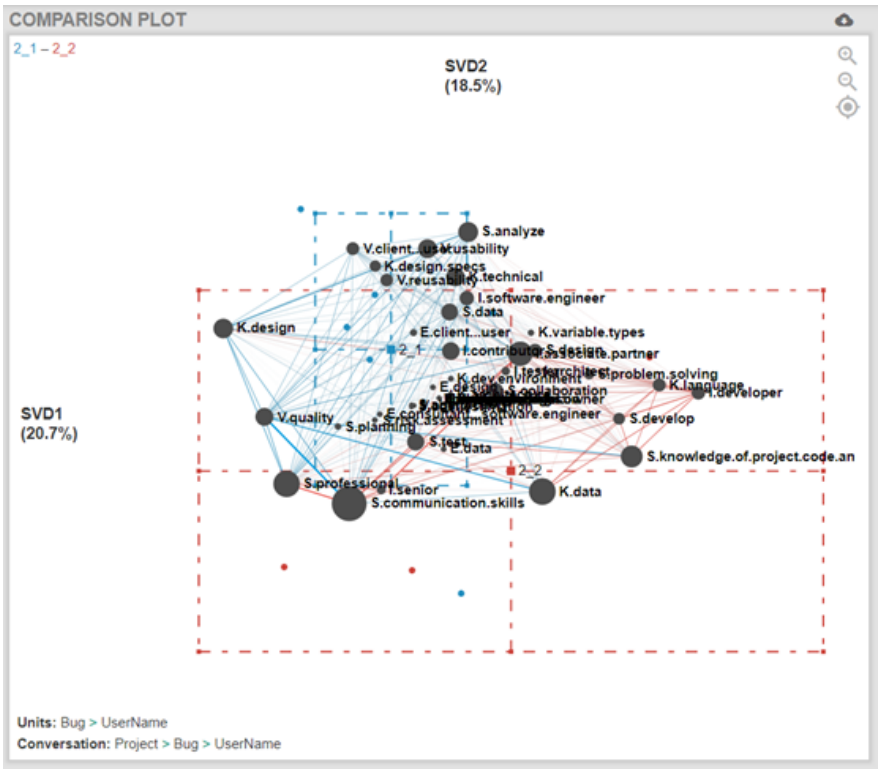


Figure 6.13: Comparison model for the two bugs of OpenOffice.

From Figure 6.13, we understand that the two networks extend into the same domains of the epistemic frame with minimal differences and similar strength in their connections. Therefore, we could say that the two bugs appear to have a similar scientific level.

6.2.3 Experiment 3: Comparisons among the conversationalists

The parameter settings for this specific experiment are the following:

- For units, the `Username` column,
- For conversation, the `Username`, `Project`, and `Bug` columns,
- For stanza window, a moving window of four lines,
- For codes, all codes from the .csv file,
- For comparison, no column.

Below, we can observe the centroids and confidence intervals for the usernames we selected to compare from LibreOffice (see Figure 6.14) and from OpenOffice (see Figure 6.15):

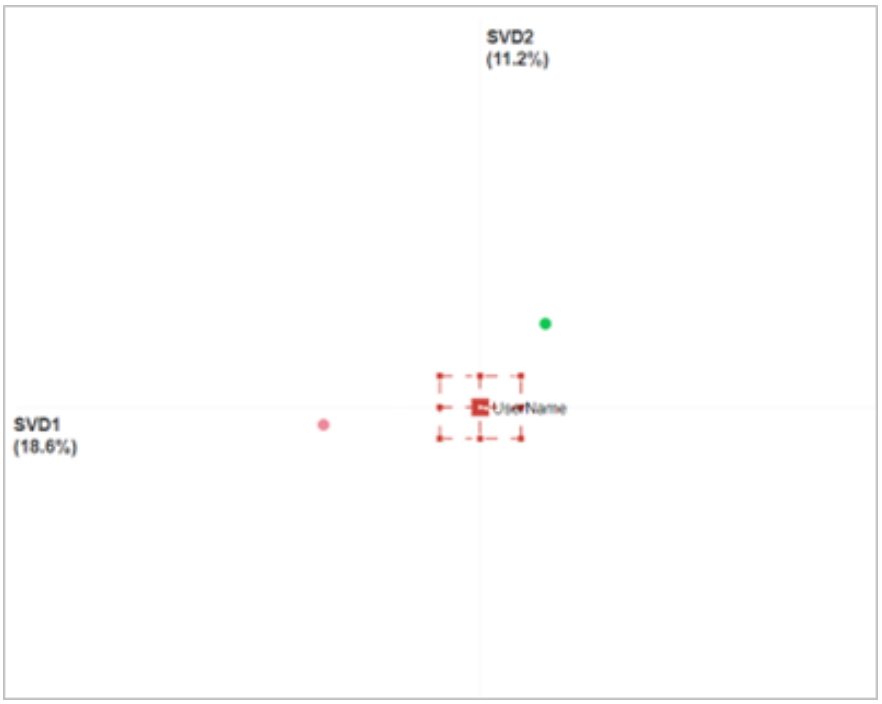


Figure 6.14: The centroids and confidence intervals for two usernames of LibreOffice (Participant_A and Participant_B).

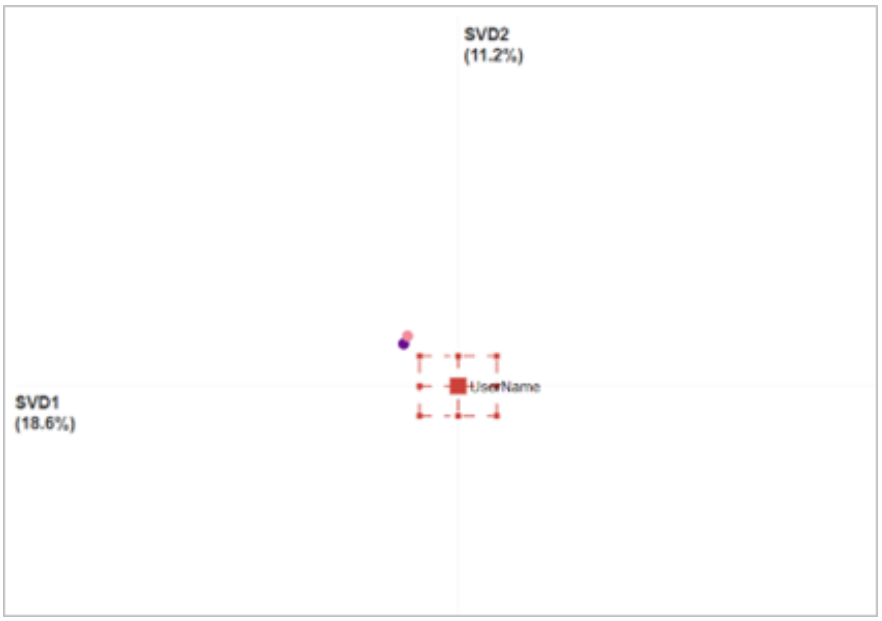


Figure 6.15: The centroids and confidence intervals for two usernames of OpenOffice (Participant_C and Participant_D).

The ENA WebKit cannot perform a statistical test for the usernames because they do not constitute separate columns in our data's .csv file. However, we can visually compare the networks. By examining the networks of the two usernames in LibreOffice (Figures 6.16 and 6.17), as well as the comparison plot (Figure 6.18), we understand that the networks are quite distinct in the ENA space.

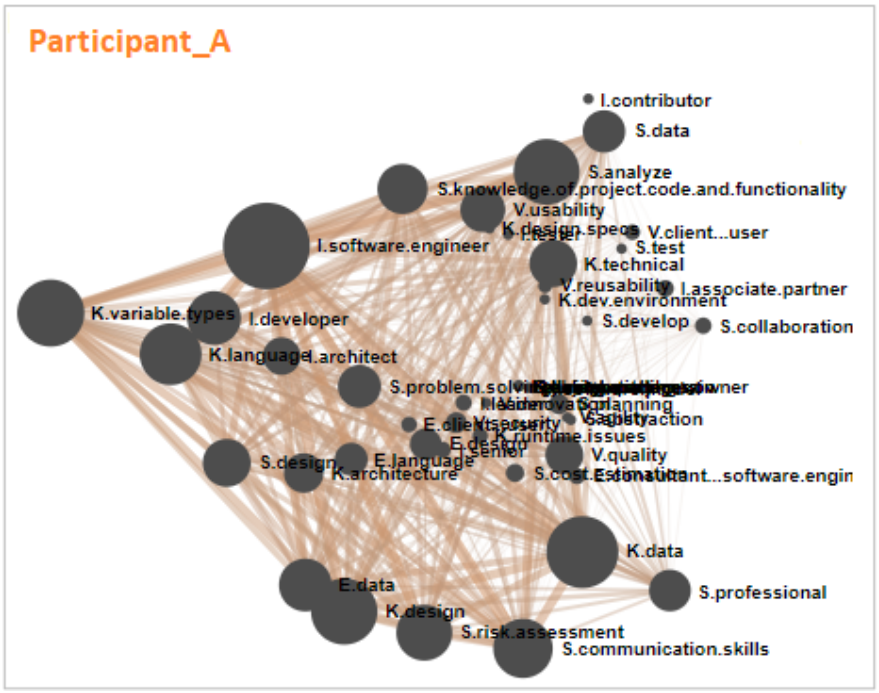


Figure 6.16: The average network for the user Participant_A from LibreOffice.

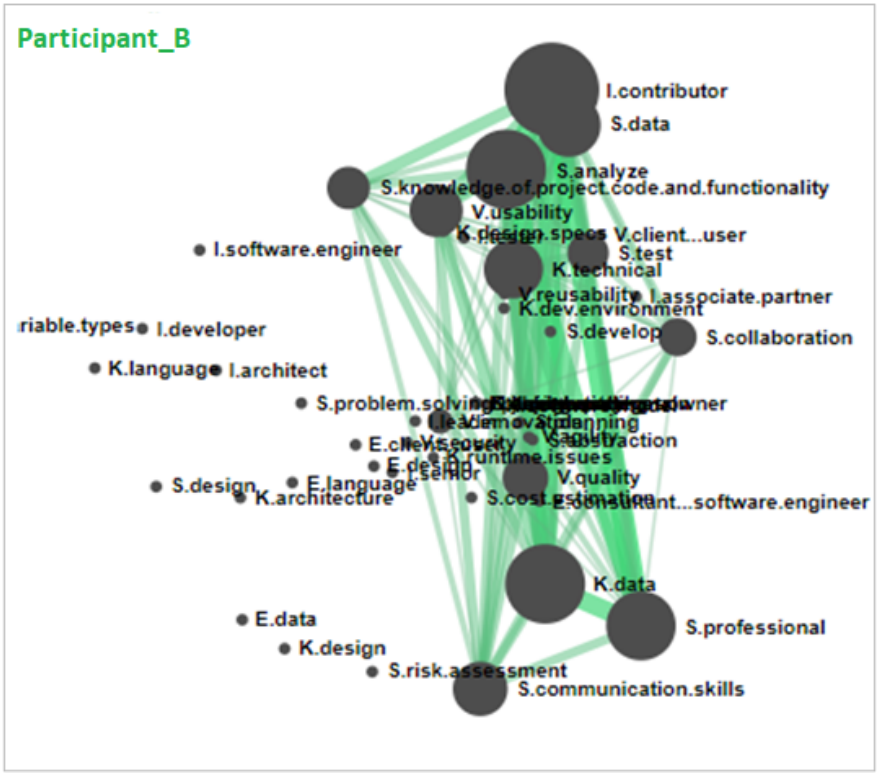


Figure 6.17: The average network for the user Participant_B from LibreOffice.

From what is apparent in the comparison plot in the ENA WebKit, Patrticipant_A had strong connections between the identity of ”software engineer” and knowledge of data, knowledge of design, communication skills, and analytical skills. On the other hand, Participant_B had stronger connections between the identity of ”contributor” and analytical skills, professionalism, knowledge of data, and technical expertise.

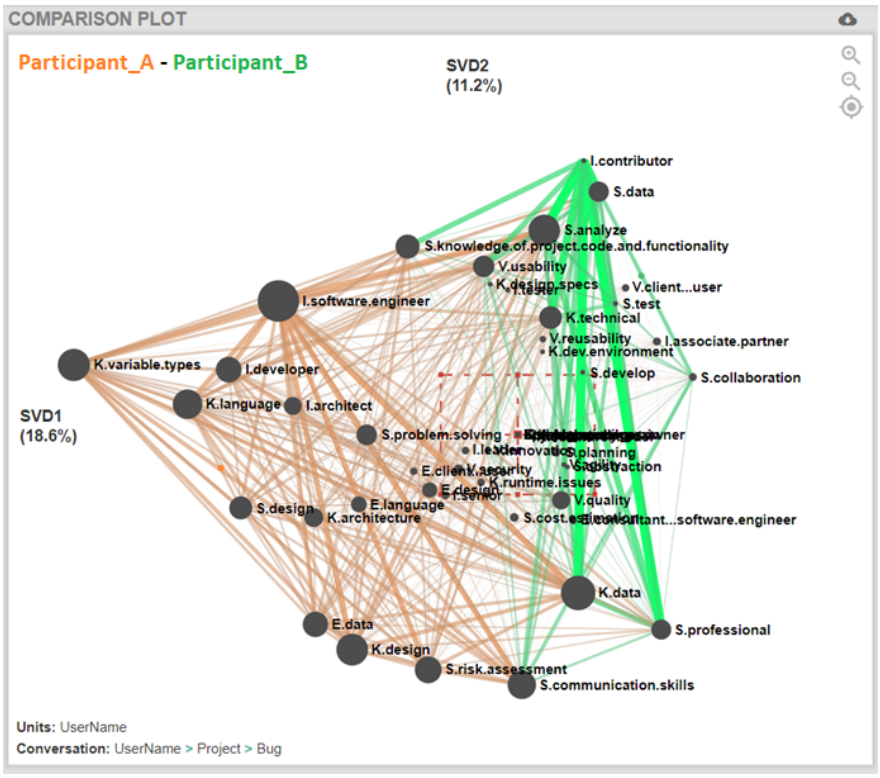


Figure 6.18: Comparison model between the two conversationalists on the LibreOffice Forum.

From Figure 6.18 we understand that the network of Patrticipant_A extends across the entire epistemic frame with stronger connections in the fields of knowledge and certain areas of skills and epistemology. In contrast, the network of Participant_B mainly extends to one side of the epistemic frame, making fewer connections to fields from different domains of the epistemic frame. We could thus say that Patrticipant_A seems to exhibit a more scientific discourse.

Looking further at the networks of the two OpenOffice usernames (Figures 6.19 and 6.20), as well as the comparison plot, we realize that the networks don’t differ significantly in the ENA space.

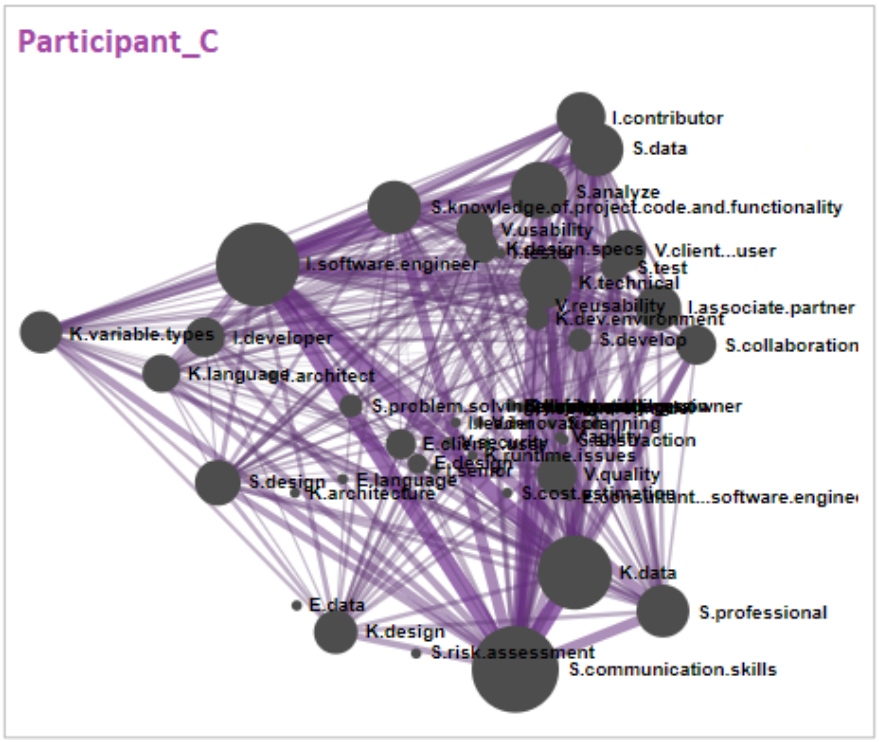


Figure 6.19: The average network for the user Participant_C from OpenOffice.

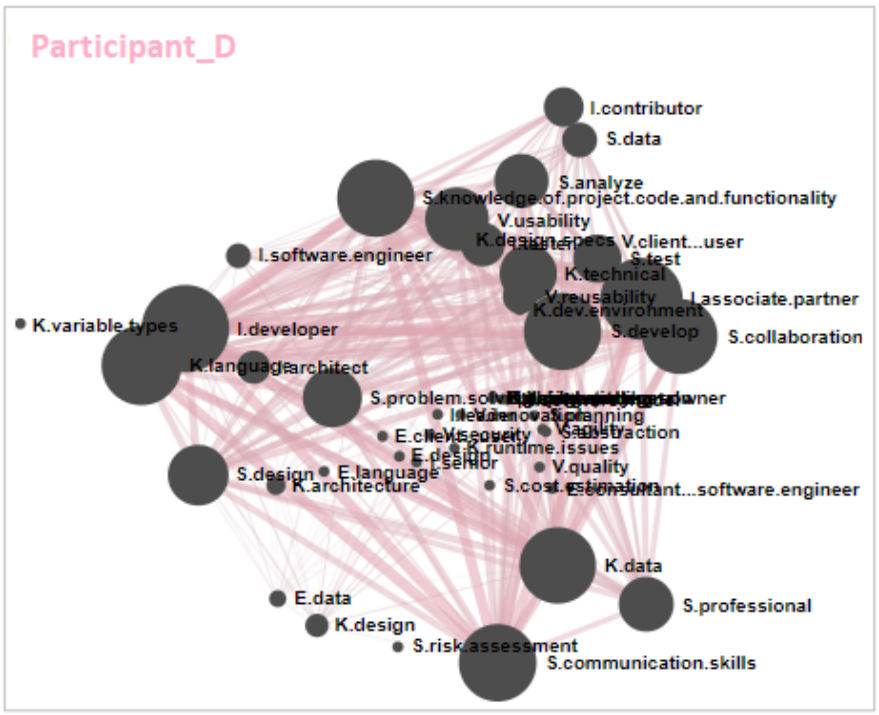


Figure 6.20: The average network for the user Participant_D from OpenOffice.

From what is apparent in the comparison plot in the ENA WebKit (Figure 6.21), Partic-

ipant_C had stronger connections between the identity of ”software engineer” and the knowledge of data, communication skills, data production and presentation skills, the identity of ”contributor,” and analytical abilities. On the other hand, Patrticipant_D had strong connections between the identity of ”developer” and the ability of code development (”develop”), as well as the identity of ”associate / partner.”

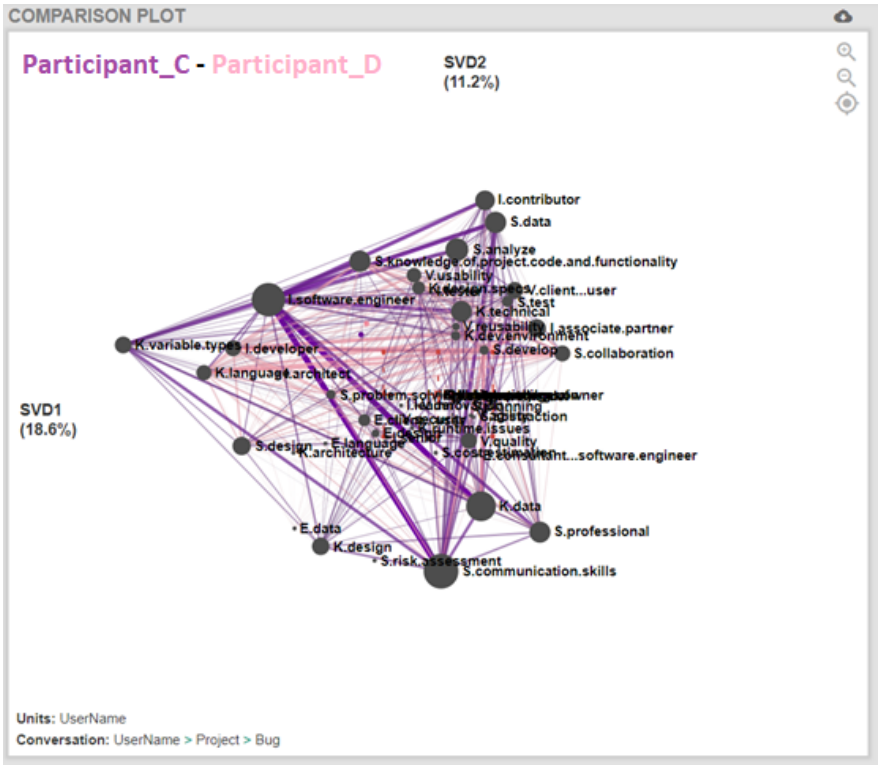


Figure 6.21: The average network for the user Patrticipant_D from OpenOffice.

From Figure 6.21, we perceive that the two networks extend throughout the entire epistemic frame. The network of Participant_C seems to have very strong connections, which is noteworthy for this specific software engineer, as the data indicates that he had fewer lines in the .csv file compared to Patrticipant_D. Therefore, we could say that both of them appear to have a similar scientific level, although Participant_C could have surpassed Patrticipant_D if he had spoken more.

NOTE: This part of our research was submitted for publication to the International Journal of Software Engineering and Knowledge Engineering by World Scientific with the title: ”Applying Epistemic Network Analysis to Open Source Software Projects” and it is currently under review.

OSS Project	Dialogue Title	Dialogue Description	URL
Open Office	Window sizes are not stored	In this dialogue, a discussion takes place regarding whether and how window sizes should be stored, so that when we reopen a file for the second time, it “remembers” the size we had set it to the first time	[11]
Open Office	setActiveSheet does not work in Hidden mode	The issue in this dialogue pertains to the fact that the API cannot be used to manipulate (e.g. export as CSV) all the sheets in a hidden mode opened Calculator file	[10]
Libre Office	o3tl::make_unsigned	In this dialogue, a “debate” takes place regarding variable types, comparisons between signed and unsigned variables, and the usage of specific methods in the project	[9]
Libre Office	LibreOffice and old Microsoft Binary file formats	The topic that concerns the participants in this specific dialogue is whether LibreOffice should continue to provide the option to save a file in the old binary format of Microsoft	[8]

Table 6.2: Dialogues selected for the ENA Analysis

	Pearson	Spearman
X Axis:	1.00	1.00
Y Axis:	1.00	1.00

Table 6.4: Pearson's and Spearman's Corellation Coefficients

7. RESILIENCE REWARDS VIA BLOCKCHAIN

The purpose of this system is not to stick to metrics that can easily be misunderstood by the user, misinterpreted, or often not fully grasped. Instead, it aims to quantify the improvement to the resilience of a project, according to the above metrics. The reward can be described as a virtual token and can act as a form of currency. In other words, if a higher - in terms of resilience - result is produced, the user receives more rewards. Specifically, each user of the application can feed the system with the open-source project they have developed, either individually or as part of a team, and after it's evaluated, reward the user based on the resilience improvement due to their work with a corresponding number of virtual tokens. The rewards they receive will now represent the resilience assessment quantitatively. To enhance security and produce an invulnerable system, the reward process is conducted via a blockchain, leveraging all the advantages mentioned earlier. Furthermore, the developed tool consists exclusively of subsystems based on open-source projects, and upon completion, it can also be classified as such.

7.1 Functional Requirements

- User registration to the system: Each user wanting to use the application should have the ability to create an account. This account should be unique, and there shouldn't be a possibility for two users to have the same registration details. With user registration, an entry in the database is created, storing all necessary details. Simultaneously, a unique account is created in the blockchain database where the user's rewards are stored.

- Linking the user account with GitHub: To evaluate the software project developed by the user, there should be a way to access their projects. These projects are, in this case, hosted on the open-source repository GitHub.

- Ability to select a project for evaluation: The user should be able to select the project they want to be evaluated from a list of all their available projects.

- Choice of evaluation axes: The user should be able to choose the evaluation axes for a specific project. These axes are predefined and may not be the same between two evaluations of the same project.

- Maintenance of user evaluation history: The user should be able to refer to previous evaluations of a project and see the rewards they earned for that specific evaluation.

- Prevent re-evaluation of the same version of a project: The system should not allow the user to evaluate a version of their project that has already been evaluated in the past.

7.2 Non-Functional Requirements

- Minimization of transaction costs on the blockchain: As any interaction of the tool with the blockchain system requires a fee, it's essential to minimize its dependence on it. Thus, adding an additional database outside the blockchain where the transaction history of a user is stored simplistically is imperative. Each evaluation constitutes a transaction. Instead of having the information about the rewards of each user only on the blockchain, where accessing and retrieving it would be costly, it's copied to the database, reducing the cost to a minimum. Additionally, the blockchain structure does not serve the purposes of a direct transaction history. This entails an increase in the system's complexity for retrieving the required information and unnecessary delays. Using the database helps keep retrieval time minimal.

- Separation of subsystems: The developed tool uses various subsystems to implement the necessary functions. Since these functions are independent of each other, the subsystems need to be clearly defined and separated. This allows the tool to be expandable and new functions to be added with great ease. It also facilitates the development stage and reduces complexity.

- Integration of subsystems: Although each subsystem is a separate entity, only as a whole can they achieve the tool's purpose. Therefore, there's a need for them to operate as a whole without increased usage complexity. This will help, as this tool is an open-source project, any developer wishing to try it without particular difficulty and under any circumstances.

7.3 Blockchain rewards

As mentioned earlier, a fundamental feature of the system is the virtual rewards. These constitute an integral part of the implementation as they expand the conventional concept of software project evaluation, a primary goal of this venture. Within this tool, these rewards are purely a measure of quality and do not possess any additional exchange utility, as one might expect, in the sense of a digital currency. This rationale is based on the fact that the evaluation of a software project, and consequently its quality, even though it can be quantified in a simpler concept, like that of currency, cannot serve as a medium for some form of exchange economy. The goal is for the developer to effortlessly, without the need to refer to a plethora of data, have an image of the quality of the project they have produced. Rewards are based on the evaluation results. Specifically, every evaluation of a project's version produces output metrics expressed as a percentage. If the project is evaluated for the first time and this percentage equals or exceeds fifty, the user will receive a token. In case a new version of an existing project is evaluated, the results of the last evaluation are compared with the new version, and the user receives a token for every improved metric. This implementation may change in the future depending on arising needs and limitations.

7.4 Architecture

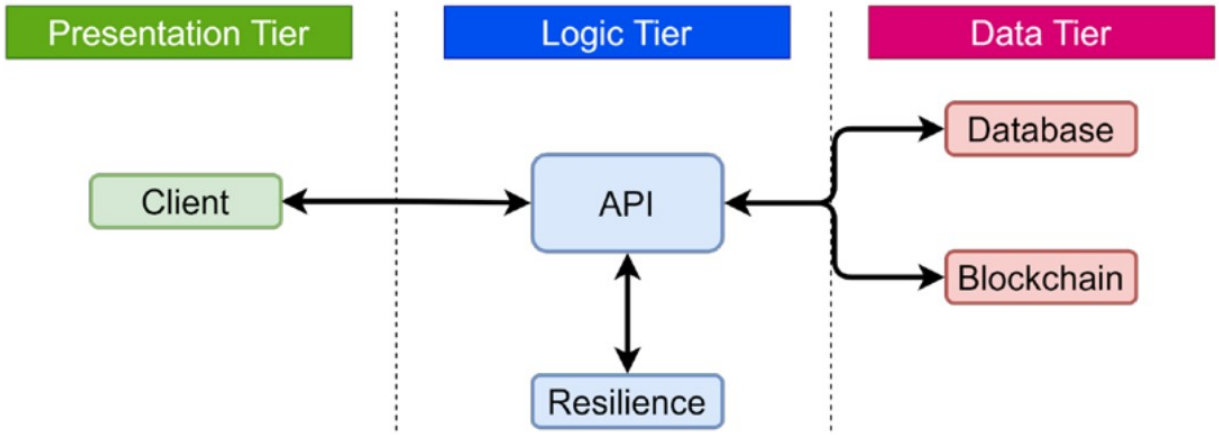
The chosen architecture for tool development is the three-tier (or multi-tier) architecture. It's a type of architecture in which the three tiers are presentation, processing, and data management. These tiers can remain independent units and are often implemented on different platforms. Generally, the independence of the tiers means that due to rapid technological advancements, an entire tier can be changed without affecting the others.

Similarly, it is possible to add an entirely new tier without the slightest change to the previous ones. In this architecture, each tier depends on its lower levels while providing services and information to its higher ones, yet without having knowledge of them. The tiers are detailed below:

Data Tier Being the most fundamental tier of the system, the Database Server provides all necessary functions for storage, retrieval, update, and maintenance of the system's data, as well as all mechanisms required for data integrity and security.

Business Logic Tier It is the main part of the software where most operations are performed, except those related to the configuration of the work screens. There's a possibility to install more than one Application Servers on different machines, leveraging any available computing power and ensuring excellent responsiveness, reliability, and scalability.

Presentation Tier The third tier of the software is the user's interface with the system (User Interface). In this tier, the management of the Work Screens (User Screens) takes place, as well as the formatting of the displayed data.



7.5 Blockchain rewards utilizing IOST

For the implementation of the blockchain, the IOST platform was chosen. IOST allows the development of new smart contracts and their immediate publication to the system.

It began as a token on the Ethereum network but eventually branched out, creating a distinct and innovative blockchain platform. Being open-source, anyone can access its implementation without restrictions. Its source code is available on GitHub. One of its advantages is the ease with which someone can implement a blockchain and tailor it to their needs. Furthermore, it has one of the fastest block creation algorithms, and its smart contracts can be developed using JavaScript. Moreover, it comes with a library also written in JavaScript, making it a suitable choice in combination with the other selected technologies. For these reasons, IOST was chosen for the implementation of this particular tool. Although the rest of the system's interface with the blockchain is done with a JavaScript library, IOST's core is written in the GO language. This makes it highly efficient as GO possesses the speed of lower-level languages, such as C and C++, but with much simpler implementation. Essentially, this subsystem is an executable in GO language that performs the implementation of a blockchain. However, for the system to serve the needs of this specific tool, it was necessary to publish a new smart contract on the blockchain. As already mentioned, in the case of IOST, smart contracts are written in JavaScript and then converted to the appropriate format to be accepted by the blockchain. For this process, the addition of Node.js to the subsystem was essential. Beyond this function, it doesn't offer anything more, but without it, the entire system wouldn't function.

At this point, it's worth noting that this specific subsystem operates in test mode. As its name suggests, all transactions taking place remain within the closed set of the developed system. This may sound restrictive, but it doesn't limit the functions and conclusions that can be drawn. Everything operates by the same rules as in the main IOST network. Upon launching the blockchain, an admin account is created in the system with a considerably large token reserve. This reserve will constitute the total rewards that can be allocated to the application users. It was considered that initially, this reserve is sufficient to cover the needs of the approach taken. The admin is also the creator of the smart contract that will govern the reward transactions in the system. This contract consists of functions to store the reward information in the space (in the sense of a database) that the contract itself has, but it doesn't allow access to this data by anyone other than its creator. This is because it was decided not to create a new, complex token type to play the role of rewards but to use the iost token itself for this purpose. Each transaction can carry a field with notes or observations that act as a comment on the transaction. The project information is stored there. Thus, every transaction contains information about which user's project it concerns, stored in the contract, and the corresponding entry is updated.

The storage of project information is done in the following manner. Each transaction is characterized by a unique alphanumeric (hash). Upon a user's registration in the system (and registration is a transaction), the updateUserEntry function is invoked, storing the user registration's hash at position zero (maxLength = 0). The first evaluation the user performs will constitute a new transaction in the system; hence, the updateUserEntry is called again, storing this transaction's hash at position one. This process is repeated for every evaluation, with the position number increasing by one each time.

For retrieving the information, the get function must first be invoked. From the result it returns, we can determine how many transactions the user has made in the system.

Subsequently, by invoking mapGet with the position of the transaction we wish to view, we have at our disposal the hash of that specific transaction, granting us access to its details, such as the reward amount or the project it pertains to. In this manner, the system administrator can access a user's information without being able to alter it.

Even though the blockchain, in this manner, retains all the information that will later be available to users, retrieval doesn't occur from it. The primary reason is that to "call" the contract and view its contents, one must perform a transaction, which incurs a cost. Furthermore, the manner in which the information is stored would require significant computational and time costs for retrieval. For this reason, parallel to the information stored on the blockchain, the portion useful to the user is also stored in the database. This way, users can easily and quickly refer indirectly to all the transactions-evaluations they have made.

7.6 Mapping resilience to rewards

This subsystem is responsible for evaluating each project according to certain metrics. The evaluation examines the resilience of each project as it emerges after specific checks. The logic behind the evaluation was implemented in an earlier time and serves as a precursor to the current tool, having provided the theoretical foundation upon which the application was developed. It is purely an open-source project and thus can be used without restrictions. More information about the Sourceographer project, and how it operates, can be found here.

Sourceographer is a software project evaluation tool that conducts specific analyses on the code of each project. Its core is written in Python, but it also incorporates libraries from PHP. Therefore, this subsystem was chosen to use Flask for its communication with the rest of the system. Flask allows for the creation of a server written in Python without introducing unnecessary complexities, unlike the widely known Django framework. However, in addition to Python, some PHP libraries are required for the evaluation. These are made available through the dependency manager, Composer. Composer is nothing more than a tool for managing (installing, updating) libraries written in PHP.

To conduct an evaluation, the user initially needs to input data into the system. This data includes a link to the GitHub project that needs to be evaluated. Subsequently, this data is sent to the subsystem, and the project's code is examined. During this examination, various parameters are calculated, which will later determine the user's rewards.

8. LIMITATIONS AND THREATS TO VALIDITY

8.1 Open Source Software Resilience Framework

In the subsequent discourse, we will delve into the potential limitations, threats to validity, and other concerns at the construct, internal, and external levels of our framework. Before implementing the OSSRF to any OSS projects, it's crucial to gauge both the maturity of the associated community and the age of the project. From a purely intuitive standpoint, we would advocate for the application of the OSSRF to projects that have demonstrated activity for a minimum duration of one (1) year and have successfully fostered a community consisting of no fewer than ten (10) contributors. Venturing into projects that don't align with these criteria, especially those that are predominantly maintained by a single entity, could yield skewed or misleading results.

We conceived the OSSRF as a nuanced adaptation of the CRF, transitioning from the realm of Urban Architecture to the more specialized arena of OSS Engineering. One of the primary objectives of the OSSRF is to chronicle and evaluate the evolving resilience of a project over time, say from one significant release to another. While our endeavors to map the original framework were driven by a comprehensive evaluation perspective of an OSS project, it's worth noting that the transposition of the two frameworks, in terms of dimension, goal, and indicator, was largely influenced by the subjective interpretations of the authors. It's essential to view this rendition of CRF to OSS as one of many potential adaptations.

In our bid to adapt the model, we incorporated indicators commonly suggested in other assessment models rooted in Software Quality. As our framework seeks to discern fluctuations at the resilience level, drawing metrics from the quality assessment domain might inadvertently introduce validity concerns. In the comprehensive rendition of the OSSRF we put forward, we've earmarked certain indicators as qualitative, warranting expert assessment, while others are of a mixed nature. For instance, they might be quantitatively measurable for object-oriented projects or require qualitative expert evaluation for non-object-oriented initiatives. In the present configuration of the OSSRF, every indicator, goal, and dimension is treated with equal significance, without any weightage. The benchmark for categorizing a project's resilience stands firm at 50%. When it comes to synthesizing the factors from the indicators to the dimensions, the process revolves around computing the mean of the corresponding factors, given their equal valuation.

A rigorous sensitivity analysis was executed for the indicators proposed in the OSSRF. Interestingly, certain indicators, such as the Testing Process (I08) which is binary in nature, exhibited marked sensitivity. There's a realm of alternative indicators that could have been incorporated, divergent from our selected set. Additionally, there are established best practices in Agile Methodologies or those linked to OSS Software Engineering, which could be pivotal in determining a project's resilience. Notable examples include continuous integration, the "release often, release early" mantra, CI/CD best practices, and privacy-centric decisions, particularly in the wake of regulations like the GDPR.

Furthermore, in our assessment of the specified six (6) projects, we harnessed specific tools to gauge certain indicators. Both commercial and open-source, these tools are duly cited, enabling interested readers to explore them in depth on their official websites. This facilitates the replication of our experiment, given the tools' accessibility either for free or as evaluation demos. Our project selection rationale was grounded in their intuitive classification as either resilient or not, aiming to ascertain the OSSRF's precision in differentiating between the two categories over time.

The qualitative metrics for these six (6) projects were pre-determined by us, and the logic underpinning these decisions is elucidated in Section 6.1.1. To further validate our choices, we engaged with five (5) industry experts for interviews. However, the limited number of participants, coupled with potential biases in their professional or demographic backgrounds, could introduce validity concerns. It's noteworthy that the tools selected for model application are exclusively developed in the PHP programming language and span five (5) distinct domains.

8.2 Metrics Aggregation

Source-o-grapher has been meticulously crafted with the primary objective of aiding users in evaluating the resilience of OSS software projects. Our ambition was to streamline the process, enabling users to glean information for a broad spectrum of indicators in a semi-automated fashion. With this vision, we seamlessly integrated Source-o-grapher with elite platforms such as Github and the esteemed PHPMetrics library. Recognizing the dynamic nature of these third-party systems and the potential for unexpected hiccups or bugs, we've woven in a manual input feature as a failsafe, ensuring uninterrupted user experience.

It's pivotal to underscore that the optimal application of Source-o-grapher's analytical prowess is best realized when directed towards Open Source Software projects that have attained a certain pedigree in terms of their longevity and the robustness of their community engagement. Venturing into projects that haven't reached this maturity could inadvertently skew the outcomes, potentially leading users astray.

Drawing inspiration from the City Resilience framework, we've tailored the Software Resilience framework specifically for the Open Source Software landscape. This intricate metamorphosis, encompassing various indicators, overarching goals, and multi-faceted dimensions, is inevitably influenced by the authors' interpretative perspectives and should be acknowledged as such.

In its current iteration, Source-o-grapher's automated functionalities are fine-tuned exclusively for the Github code repository. Furthermore, it resonates best with OSS projects primarily scripted in the PHP language. On the operating system front, Source-o-grapher has been specifically engineered for, and rigorously tested on, the Ubuntu Linux Operating System, ensuring optimal performance within this environment.

8.3 Epistemic Network Analysis

In the course of this research, we have undertaken a detailed exploration through three (3) distinct experiments, each one shedding light on the potential of ENA as a potent assessment tool within the software engineering realm. These experiments specifically focus on dialogues and their underlying epistemological foundations. In this segment, it's imperative to elucidate certain aspects which we recognize as potential threats to the validity of our manuscript.

The ENA methodology, central to our study, has been applied to two (2) specific projects: OpenOffice and LibreOffice. Both these entities operate within the office suite domain. It's noteworthy that LibreOffice emerged as a primary successor, branching out as a forked OSS project from its predecessor, OpenOffice. The selection of bugs for our investigation was limited, encompassing a total of four (4), split evenly between the two aforementioned projects. Furthermore, the participants chosen for these experiments are not random but were handpicked based on specific criteria. Our choice of tool for conducting the ENA analysis was the ENA WebKit. An additional dimension to consider is that the codes employed in the dialogues were not objectively derived but were influenced by the authors' interpretative perspectives.

Considering all the outlined parameters and specificities, one must exercise caution while extrapolating our findings. This research, in its essence, should be perceived as an initial foray into this domain, producing results that are both indicative and preliminary. Nevertheless, our findings are emblematic of the potential of epistemic network analysis, illustrating its capability to yield quantifiable, insightful outcomes when delving into the epistemic content of software projects.

9. CONCLUSIONS AND FUTURE WORK

9.1 Open Source Software Resilience Framework

In this research endeavor, we are embarked on a journey to adapt the concept of Urban Resilience into the realm of Open Source Software (OSS). By understanding resilience, especially within dynamic systems like OSS projects that frequently encounter various stresses and crises, we aim to dissect the influence of these challenges on a project's longevity and survival. Our proposed framework, termed OSSRF, emerges as a robust evaluation methodology, anchoring its foundations in offering a comprehensive theoretical foundation for OSS assessment. We've meticulously applied the OSSRF model to a total of six (6) open source ventures. Intriguingly, half of these, three (3) to be precise, are instinctively viewed as non-resilient, while the remaining three (3) exude resilience. At its core, OSSRF is conceptualized as a resilience-centric model, honed in on gauging the resilience of an OSS project as it undergoes metamorphosis over its lifecycle, transitioning from one version to the next. Positioned in this unique context, we firmly believe that OSSRF stands as an alternative paradigm for evaluating OSS projects. It's imperative to note that OSSRF and other established Quality Assessment Models for OSS aren't necessarily in opposition and shouldn't be perceived as mutually exclusive.

An examination of our application results reveals a compelling narrative. Both resilient and non-resilient projects are aptly assessed as such by OSSRF. The projects exuding resilience consistently register elevated scores, particularly in the realms of Business, Legal, and Community. This outcome underscores an interesting insight: the more vibrant and active a project is, the higher its organizational proficiency, spanning facets like licensing protocols, codes of conduct, contribution guidelines, and beyond. Conversely, projects leaning towards non-resilience showcase diminished scores predominantly in Business & Legal and the Social (Community) spectrum. A plausible hypothesis for this trend could be the project's origins as a solo endeavor, devoid of community allure, or perhaps an absence of an initial vision geared towards sustainability or commercial appeal. Eventually, such projects witnessed a gradual decline, culminating in their obsolescence.

Among the non-resilient projects, an intriguing observation arose. Our framework adeptly identified several projects as non-resilient, spanning from those with limited lifespans or contributor bases to projects like PHPExcel. The latter, for a significant duration (primarily in its initial lifecycle), garnered substantial contributors and exhibited extended vitality before reaching its archival phase. We postulate that OSSRF's unique approach, closely monitoring consecutive project releases, endows it with the nimbleness to detect resilience downturns, correlating them with specific model dimensions.

As we gaze into the future, we aspire to empirically evaluate the qualitative indicators embedded in our current framework, leveraging a more expansive cohort of independent OSS connoisseurs. Furthermore, we're intrigued to discern if domain-centric experts perceive these qualitative indicators through a different lens. Our quest also encompasses probing the influence of various factors like code repositories, programming languages,

project age, specific domains, or even developmental ideologies on our chosen indicators. Such endeavors present a golden chance to contemplate integrating weights at the indicators, goals, or dimension tiers. We're also keen on experimenting with alternative indicators, refining the sensitivity analysis integral to our model.

At this juncture, our model integrates a mix of indicators. Our forward-looking vision encompasses crafting two distinct model iterations: one laser-focused on Object-Oriented Programming (OOP) OSS projects, and the other geared towards non-OOP OSS projects. Broadly speaking, our fervent motivation revolves around exploring model variants optimized for specific OSS application territories, programming dialects, or even developmental stacks. Additionally, we're eager to implement OSSRF on projects that have grappled with specific challenges or stresses over their lifespan. Our mission will be to discern the impact of these crises on a project's resilience quotient. We also harbor ambitions to classify and systematically organize the stresses and crises inherent to OSS projects, possibly through structured mechanisms like taxonomies.

On the technological frontier, we're diligently developing solutions that promise to semi-automate our assessment procedure. Such advancements aim to empower stakeholders, facilitating their experimental endeavors related to OSS project resilience. And lastly, one of our imminent objectives is to actively solicit feedback from influential figures within the global OSS community. We deeply value insights from developers, academics, stakeholders, project managers, and proprietors of OSS-centric enterprises regarding the potential and efficacy of OSSRF.

9.2 Metrics Aggregation

Within the scope of this research, we introduce "Source-o-grapher," a sophisticated tool crafted specifically for the meticulous assessment of OSS projects, primarily focusing on the resilience dimension. Our conviction is rooted in the notion that a tangible parallel exists between software resilience and the holistic health of software. This is primarily because the myriad of challenges or crises an OSS project encounters frequently reverberate through its foundational pillars. These pillars encompass its vibrant community, the structural integrity of its source code, its long-term sustainability prospects, and a host of other integral facets.

The Source-o-grapher is not just another tool; it's an embodiment of our vision. It is architecturally engineered to function both as an incisive analyzer, leveraging the Software Resilience Framework for OSS, and simultaneously as an assistant that facilitates the seamless collation of all pertinent information related to the indicators essential for the underlying analysis. Keeping future scalability and integration in mind, our design philosophy for Source-o-grapher emphasizes effortless compatibility with third-party systems, be it other analytical tools or diverse source code repositories.

Peering into the horizon, our ambitions for Source-o-grapher are vast and multi-faceted. We are driven to expand its repertoire to embrace a broader spectrum of source code

repositories, transcending its current affinity with Github. Our roadmap includes integrating functionalities that encompass platforms like GitLab. Beyond that, our aspiration is to craft specialized ports tailored for private GitLab repositories. Such an endeavor would be a game-changer, empowering inner-source corporate entities and organizations to conduct thorough assessments of their in-house projects. We are staunch advocates of the idea that such enhancements can revolutionize the way companies, academic institutions, and diverse organizations approach the assessment of OSS projects, leveraging the unparalleled insights provided by the Resilience Framework for OSS.

Our vision doesn't stop there. We are on a relentless quest to amplify our array of analyzers to encompass a wider range of programming languages, including but not limited to, stalwarts like Java and Python.

In our continuous effort to bolster the capabilities of Source-o-grapher, we have embarked on an exciting journey to integrate it with the renowned Grimoire Lab tool. Our motivation is to harness some of its myriad groundbreaking features, notably the ”parceval parser” tailored for OSS projects. We are bubbling with optimism that this synergistic integration will seamlessly weave into Source-o-grapher's upcoming iteration.

To round off our vision, one of our cornerstone objectives is to devise mechanisms that facilitate the automatic assessment of a sequential series of OSS software versions. This, we believe, will be complemented perfectly with the introduction of innovative visualizations. Such visual tools will meticulously chronicle the evolutionary trajectory of resilience facets inherent to OSS software, providing stakeholders with a temporal lens into the software's resilience journey.

9.3 Blockchain rewards

Having reached the completion of this specific tool, there's an absolute satisfaction with the results of the effort and the goals that were set. However, looking ahead, there are areas that could be improved, allowing us to have a ready-to-use tool that can be immediately leveraged. The ability to expand and enhance is already present, so it's just a matter of setting more ambitious goals we'd like to achieve. These objectives can be divided into improving the existing system and adding new functionalities.

The first category of goals stems from the development and testing process of the tool. With the current separation, there has been almost complete independence of the system's functions achieved. The next logical step in this direction would be to split the API and GUI subsystems into two distinct entities, each with its unique properties. Furthermore, it would be prudent to reduce the size of each image produced by Docker by using a smaller and lighter software base, allowing the application to operate even more efficiently. In the realm of improvement, since the tool is a web application that will engage multiple users simultaneously, it would be beneficial to add a load and request management tool, like nginx or apache.

Turning to the second category of objectives, it encompasses ideas that weren't realized

but also new ones that emerged during the development phase. Initially, the evaluations are exclusively for GitHub projects written in the C or PHP languages. We'd like to accommodate other open-source repositories and have the capability to evaluate all significant languages, offering the potential to extend the options available. In terms of the user interface, we aim to enrich the available information without making it redundant. This could include graphics or tables of evaluation results, enhancing the user experience and providing them with all the generated information for their assessment as they see fit.

Furthermore, as technology evolves, it's essential to ensure that the tool remains adaptable and flexible. Integrating feedback mechanisms within the tool could be invaluable, allowing users to provide direct insights on areas of improvement. This would ensure that the tool not only meets but anticipates the needs of its users, paving the way for a more collaborative and interactive platform. Future iterations of the tool could also focus on enhancing security features, ensuring data integrity, and promoting a more user-friendly environment. The journey might be ongoing, but with these steps, the path to achieving greater success becomes clearer.

9.4 Epistemic Network Analysis

The proliferation of educational tools and the burgeoning reservoir of resources over the past few years has underscored the imperative to critically assess their inherent value. This necessitates a meticulous selection process, tailored to the nuanced requirements of specific scenarios. Through the lens of Epistemic Network Analysis (ENA), we venture into the realm of log files generated from diverse ”educational” endeavors. These logs are adeptly encoded to register the emergence and prominence of salient fields within a domain, a case in point being software engineering, which was the epicenter of our analytical exploration. Delving deeper, for any duo of elements nestled within this domain, we mathematically discern the potency of their interlinkage in a structured scientific network. This quantification is predicated upon the periodicity of their mutual manifestation within the wealth of log file data.

Our investigative journey illuminated the widespread applicability of ENA across a kaleidoscope of scientific terrains, be it the intricate world of ethnography or the expansive universe of education. Amidst this vast expanse of potential ENA applications, our research endeavor was laser-focused on dissecting the scientific underpinnings of software engineering. Armed with this analytical framework, we juxtaposed scientific networks emanating from specific dialogues, rendering a panoramic view of their inherent characteristics. By delineating the scientific temperament of a dialogue, we are empowered to extrapolate overarching insights about the ethos of a project's community. This, in turn, paves the way for informed decisions when navigating the labyrinth of open source projects, thereby fostering a conducive environment for knowledge assimilation.

The primary thrust of this manuscript was to unfurl the intricate tapestry of ENA's theoretical foundations and its multifaceted applications. Venturing further, we orchestrated a series of experiments, meticulously scrutinizing dialogues emanating from two iconic open

source projects: LibreOffice and OpenOffice. Our objective was to gauge their epistemological underpinnings. When viewed through a macroscopic lens, LibreOffice emerged with a denser fabric of scientific dialogues. However, OpenOffice was nipping at its heels. It's paramount to acknowledge that such insights should be absorbed with a grain of salt. Given the circumscribed scope of our sample, these findings don't necessarily paint a holistic picture of the broader communities enveloping both projects. Nevertheless, this exploration serves as a foundational blueprint, a precursor to more expansive endeavors.

Envisioning the trajectory of future research, there's a tantalizing prospect of delving into a more diverse mosaic of communities. This would facilitate a more nuanced and comprehensive understanding of the epistemological underpinnings inherent in each. Our experimental canvas could be broadened to embrace a myriad of domains within the software engineering paradigm, such as intricate databases, state-of-the-art customer relationship management systems, and avant-garde communication platforms. This could be further accentuated by dissecting applications sculpted in eclectic programming languages or those rooted in divergent developmental frameworks, encompassing projects that span a spectrum of maturity milestones, be it in chronological age or the vibrancy of their communities.

From an analytical perspective, our gaze could extend to juxtaposing dialogues between the cerebral world of programmers and other pivotal contributors who don't necessarily dabble in programming. A particularly intriguing avenue to traverse would be the comparative ENA analysis of dialogues orbiting around bugs vis-à-vis those centered on enhancement features.

The allure of Large Language Models (LLMs) is undeniable. With their inherent prowess to counteract challenges like the deluge of information and ingrained cognitive biases that potentially skew traditional manual analyses, they emerge as potent candidates for automation in epistemic evaluation. Thus, channeling such an LLM into the intricate matrix of Open Source Software Epistemic analysis represents a tantalizing challenge, brimming with research potential.

To cap it off, forthcoming research endeavors might delve deeper, orchestrating a more stringent and robust analysis of dialogues. This could encompass a broader array of codes and would be helmed by domain specialists, ensuring unparalleled precision and depth.

ABBREVIATIONS - ACRONYMS

CRF	City Resilience Framework
ENA	Epistemic Analysis Network
OSS	Open Source Software

REFERENCES

[1] Best frameworks for 2021 by Kinsta. https://kinsta.com/blog/php-frameworks/. [Online].

[2] C Coverage Test Tool - Website. http://semdesigns.com/Products/TestCoverage/CTestCoverage.html. [Online].

[3] CHAOSS Metrics - Starter Project Health. https://chaoss.community/kb/metrics-model-starter-project-health/. [Online].

[4] Composer - Official Website. https://https://getcomposer.org/. [Online].

[5] Composer on Github. https://github.com/composer/composer. [Online].

[6] Core-js - Founders post about core-js facing a sustanability crisis. https://github.com/zloirock/core-js/blob/master/docs/2023-02-14-so-whats-next.md. [Online].

[7] Core-js - Github: Contributors Insights Statistics. https://github.com/zloirock/core-js/graphs/contributors. [Online].

[8] Dialogue: LibreOffice and old Microsoft Binary file formats. http://document-foundation-mail-archive.969070.n3.nabble.com/LibreOffice-and-old-Microsoft-Binary-file-formats-td4272510.html. [Online].

[9] Dialogue: o3tl::make unsigned. http://document-foundation-mail-archive.969070.n3.nabble.com/o3tl-make-unsigned-td4272980.html. [Online].

[10] Dialogue: setActiveSheet does not work in Hidden mode. https://bz.apache.org/ooo/show_bug.cgi?id=127395. [Online].

[11] Dialogue: Window sizes are not stored. https://bz.apache.org/ooo/show_bug.cgi?id=91768. [Online].

[12] ENA WebKit. http://www.epistemicnetwork.org/. [Online].

[13] Gitstats Tool - Official Website. http://gitstats.sourceforge.net/. [Online].

[14] GNU Emacs Manuals. https://www.gnu.org/software/emacs/manual/. [Online].

[15] How Does Open Source Die?, O'Reilly. https://www.oreilly.com/library/view/open-source-for/0596101198/ch01s07.html. [Online].

[16] Laravel - Official Website. https://laravel.com/. [Online].

[17] Laravel on Github. https://github.com/laravel/laravel/. [Online].

[18] Linux Kernel Github Repository. https://github.com/torvalds/linux. [Online].

[19] Linux Kernel Official Website. https://www.kernel.org. [Online].

[20] OKapi Github Repository. https://github.com/liip/Okapi. [Online].

[21] PHPCoverage Tool - Website. http://phpcoverage.sourceforge.net/. [Online].

[22] PHPExcel Github Repository. https://github.com/PHPOffice/PHPExcel. [Online].

[23] PHPMyAdmin - Official Website. https://www.phpmyadmin.net/. [Online].

[24] PHPMyAdmin on Github. https://github.com/phpmyadmin/phpmyadmin. [Online].

[25] PHPQA Tool, Official website. https://edgedesigncz.github.io/phpqa/. [Online].

[26] RedHat - A guide to open source project governance models. https://www.redhat.com/en/resources/guide-to-open-source-project-governance-models-overview/. [Online].

[27] SciTools Understand - Website. https://scitools.com/features/. [Online].

[28] 100 Resilient Cities. http://www.100resilientcities.org/, 2013. [Online].

[29] A. Ampatzoglou, A. Gkortzis, S. Charalampidou, and P. Avgeriou. An embedded multiple-case study on oss design quality assessment across domains. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 255–258, Oct 2013.

[30] Sandro Andrade and Filipe Saraiva. Principled evaluation of strengths and weaknesses in floss communities: A systematic mixed methods maturity model approach. In *IFIP International Conference on Open Source Systems*, pages 34–46. Springer, 2017.

[31] C Warren Axelrod. Investing in software resiliency. 2009.

[32] I Baxter. Branch coverage for arbitrary languages made easy: Transformation systems to the rescue. *IW APA TV2/IC SE2001. http://techwell.com/sites/default/files/articles/XUS1173972file1_0. pdf*, 2001.

[33] Benjamin Birkinbine. Conflict in the commons: Towards a political economy of corporate involvement in free and open source software. *The Political Economy of Communication*, 2(2), 2015.

[34] Benjamin J Birkinbine. Conflict in the commons: Towards a political economy of corporate involvement in free and open source software. *The Political Economy of Communication*, 2(2), 2015.

[35] P. Bourque and R.E. Fairley, editors. *Guide to the Software Engineering Body of Knowledge, Version 3.0*. IEEE Computer Society, 2014.

[36] Michael Bray, Kimberly Brune, David A Fisher, John Foreman, and Mark Gerken. C4 software technology reference guide-a prototype. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1997.

[37] Simon Butler, Jonas Gamalielsson, Björn Lundell, Per Jonsson, Johan Sjöberg, Anders Mattsson, Niklas Rickö, Tomas Gustavsson, Jonas Feist, Stefan Landemoo, et al. An investigation of work practices used by companies making contributions to established oss projects. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 201–210. ACM, 2018.

[38] Javier Luis Cánovas Izquierdo and Jordi Cabot. Enabling the definition and enforcement of governance rules in open source systems. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, pages 505–514. IEEE Press, 2015.

[39] Kathy Charmaz. *Constructing grounded theory : a practical guide through qualitative analysis*. Sage Publications, London; Thousand Oaks, Calif., 2006.

[40] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.

[41] J Da Silva and B Morera. City resilience framework. *Arup & Rockefeller Foundation. Online: http://publications. arup. com/Publications/C/City_Resilience_Framework. aspx [12/15/2015]*, 2014.

[42] Carlo Daffara. The sme guide to open source software. *recuperado el*, 1, 2009.

[43] Organización Internacional de Normalización. *ISO-IEC 25010: 2011 Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation (SQuaRE)-System and Software Quality Models*. ISO, 2011.

[44] F. Umm e Laila, S. Najeed Ahmed Khan, and T. Asad Arfeen. Framework for identification of critical factors for open source software adoption decision in mission-critical it infrastructure services. *IETE Journal of Research*, 0(0):1–14, 2021.

[45] European Commission. OPEN SOURCE SOFTWARE STRATEGY 2020 –2023 Think Open. `https://ec.europa.eu/info/departments/informatics/open-source-software-strategy_en/`, 2020. [Online. Retrieved: September 11, 2022].

[46] Slinger J. Fang, H. A Systematic Literature Review on Trust in the Software Ecosystem. In *Empirical Software Engineering*, 2022.

[47] Kecia AM Ferreira, Mariza AS Bigonha, Roberto S Bigonha, Luiz FO Mendes, and Heitor C Almeida. Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software*, 85(2):244–257, 2012.

[48] Jonas Gamalielsson and Björn Lundell. ”Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved?”. *Journal of Systems and Software*, 89:128 – 145, 2014.

[49] Ross Gardler and Gabriel Hanganu. Governance models. *Open Source Software Watch, last modified February*, 14, 2012.

[50] James Paul Gee. *An Introduction to Discourse Analysis: Theory and Method (4th ed.)*. Routledge, 2014.

[51] Clifford Geertz. *Thick description*. 1973.

[52] B Glaser. Glaser and strauss: Developing grounded theory. 1967.

[53] Charles Goodwin. Professional vision. *American Anthropologist*, 96(3):606–633, 1994.

[54] Kim Herzig, Sascha Just, and Andreas Zeller. It’s not a bug, it’s a feature: How misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE ’13, pages 392–401, Piscataway, NJ, USA, 2013. IEEE Press.

[55] City Resilience Index. City resilience framework. *The Rockefeller Foundation and ARUP*, 2014.

[56] Javier Luis Cánovas Izquierdo and Jordi Cabot. The role of foundations in open source projects. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Society*, ICSE-SEIS ’18, pages 3–12, New York, NY, USA, 2018. ACM.

[57] Slinger Jansen. Measuring the health of open source software ecosystems: Beyond the scope of project health. *Information and Software Technology*, 56(11):1508–1519, 2014. Special issue on Software Ecosystems.

[58] Slinger Jansen. Measuring the health of open source software ecosystems: Beyond the scope of project health. *Information and Software Technology*, 56(11):1508–1519, 2014.

[59] Ann C Klassen, John Creswell, Vicki L Plano Clark, Katherine Clegg Smith, and Helen I Meissner. Best practices in mixed methods for quality of life research. *Quality of life Research*, 21:377–380, 2012.

[60] Apostolos Kritikos and Ioannis Stamelos. Open source software resilience framework. In *IFIP International Conference on Open Source Systems*, pages 39–49. Springer, 2018.

[61] Taibi D. Tosi D. Lavazza L. Morasca S Lenarduzzi, V. Open Source Software Evaluation, Selection, and Adoption: a Systematic Literature Review. In *Proc. of 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 437–444, 2020.

[62] V. Lenarduzzi, D. Taibi, D. Tosi, L. Lavazza, and S. Morasca. Open source software evaluation, selection, and adoption: a systematic literature review. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 437–444, 2020.

[63] Juho Lindman, Anna Paajanen, and Matti Rossi. Choosing an open source software license in commercial context: A managerial perspective. In *36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2010)*, pages 237–244. IEEE, 2010.

[64] Lech Madeyski. *Test-driven development: An empirical evaluation of agile practice*. Springer Science & Business Media, 2009.

[65] Hanna Mäenpää, Simo Mäkinen, Terhi Kilamo, Tommi Mikkonen, Tomi Männistö, and Paavo Ritala. Organizing for openness: six models for developer involvement in hybrid oss projects. *Journal of Internet Services and Applications*, 9(1):17, 2018.

[66] Robert Martin. Oo design quality metrics. *An analysis of dependencies*, 12:151–170, 1994.

[67] Dharmesh Thakker Max Schireson. The Money In Open-Source Software. `https://techcrunch.com/2016/02/09/the-money-in-open-source-software/`, 2016. [Online].

[68] Joseph A Maxwell et al. *Designing a qualitative study*, volume 2. The SAGE handbook of applied social research methods, 2008.

[69] Jeff McAffer. Microsoft joins the Open Source Initiative. `https://open.microsoft.com/2017/09/26/microsoft-joins-open-source-initiative/`, 2017. [Online].

[70] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, Dec 1976.

[71] Robert Campbell McColl, David Ediger, Jason Poovey, Dan Campbell, and David A Bader. A performance evaluation of open source graph databases. In *Proceedings of the first workshop on Parallel programming for analytics applications*, pages 11–18. ACM, 2014.

[72] Jens Meinicke, Chu-Pan Wong, Christian Kästner, Thomas Thüm, and Gunter Saake. On essential configuration complexity: Measuring interactions in highly-configurable systems. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE 2016, pages 483–494, New York, NY, USA, 2016. ACM.

[73] Microsoft Corporation. Microsoft acquires Github. `https://news.microsoft.com/announcement/microsoft-acquires-github/`, 2018. [Online. Retrieved: September 11, 2022].

[74] Vishal Midha and Prashant Palvia. Factors affecting the success of open source software. *Journal of Systems and Software*, 85(4):895–905, 2012.

[75] José P Miguel, David Mauricio, and Glen Rodríguez. A review of software quality models for the evaluation of software products. *arXiv preprint arXiv:1412.2977*, 2014.

[76] Vision Mobile. Open governance index-measuring the true openness of open source projects from android to webkit. 2011.

[77] Neeshal Munga, Thomas Fogwill, and Quentin Williams. The adoption of open source software in business models: a red hat and ibm case study. In *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, pages 112–121. ACM, 2009.

[78] Roy D Pea. Practices of distributed intelligence and designs for education. *Distributed cognitions: Psychological and educational considerations*, 11:47–87, 1993.

[79] James Piggot and Chintan Amrit. How healthy is my project? open source project attributes as indicators of success. In *IFIP International Conference on Open Source Systems*, pages 30–44. Springer, 2013.

[80] Karl Michael Popp. *Best Practices for commercial use of open source software: Business models, processes and tools for managing open source software*. BoD–Books on Demand, 2015.

[81] Eric Raymond. The cathedral and the bazaar. *Philosophy & Technology*, 12(3):23, 1999.

[82] Dan Remenyi. *Grounded theory - the reader series*. Acpil, 2 edition, July 2014.

[83] Gregorio Robles, Jesus M Gonzalez-Barahona, and Israel Herraiz. An empirical approach to software archaeology. In *Proc. of 21st Int. Conf. on Software Maintenance (ICSM 2005), Budapest, Hungary*, pages 47–50, 2005.

[84] I Sam Saguy and Vera Sirotinskaya. Challenges in exploiting open innovation's full potential in the food industry with a focus on small and medium enterprises (smes). *Trends in Food Science & Technology*, 38(2):136–148, 2014.

[85] Ioannis Samoladas, Georgios Gousios, Diomidis Spinellis, and Ioannis Stamelos. The sqo-oss quality model: measurement based open source software evaluation. In *IFIP International Conference on Open Source Systems*, pages 237–248. Springer, 2008.

[86] D Shaffer. Transforming big data into meaningful insights: Introducing quantitative ethnography. *Scientia*, 2018.

[87] D Shaffer and A Ruis. Epistemic network analysis: A worked example of theory-based learning analytics. *Handbook of learning analytics*, 2017.

[88] David Williamson Shaffer. Epistemic network analysis: Understanding learning by using big data for thick description. In *International handbook of the learning sciences*, pages 520–531. Routledge, 2018.

[89] Abraham Silberschatz, Greg Gagne, and Peter B Galvin. *Operating system concepts*. Wiley, 2018.

[90] Chandrasekar Subramaniam, Ravi Sen, and Matthew L Nelson. Determinants of open source software project success: A longitudinal study. *Decision Support Systems*, 46(2):576–585, 2009.

[91] Daniel D Suthers and Caterina Desiato. Exposing chat features through analysis of uptake between contributions. In *2012 45th Hawaii international conference on system sciences*, pages 3368–3377. IEEE, 2012.

[92] Jose Teixeira, Gregorio Robles, and Jesús M González-Barahona. Lessons learned from applying social network analysis on an industrial free/libre/open source software ecosystem. *Journal of Internet Services and Applications*, 6(1):14, 2015.

[93] A. Arfeen A. Y. Ali M. Khurram U. Laila, N. Ahmed and M. A. Khan. Mission-critical open-source software adoption model validation using partial least square - structural equation modeling. *J. Softw. Evol. Process*, 35(2):e2514, 2023.

[94] M Välimäki and Ville Oksanen. Evaluation of open source licensing models for a company developing mass market software. *Law and Technology*, 2002.

[95] Mikko Valimaki. Dual licensing in open source software industry. 2002.

[96] Robert Viseur. Identifying success factors for the mozilla project. In *IFIP International Conference on Open Source Systems*, pages 45–60. Springer, 2013.

[97] Anthony Wasserman, Murugan Pal, and Christopher Chan. The business readiness rating model: an evaluation framework for open source. In *Proceedings of the EFOSS Workshop, Como, Italy*, 2006.

[98] Anthony I Wasserman, Xianzheng Guo, Blake McMillian, Kai Qian, Ming-Yu Wei, and Qian Xu. Osspal: Finding and evaluating open source software. In *IFIP International Conference on Open Source Systems*, pages 193–203. Springer, 2017.

[99] Arthur Henry Watson, Dolores R Wallace, and Thomas J McCabe. *Structured testing: A testing methodology using the cyclomatic complexity metric*, volume 500. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1996.

[100] Steve Weber. *The success of open source*. Harvard University Press, 2004.

[101] David A Wheeler. More than a gigabuck: Estimating gnu/linux's size, 2001.

[102] Andreas Wieland and Carl Marcus Wallenburg. The influence of relational competencies on supply chain resilience: a relational view. *International Journal of Physical Distribution & Logistics Management*, 43(4):300–320, 2013.