Автономная некоммерческая организация высшего образования «Университет Иннополис» (АНО ВО «Университет Иннополис»)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)
по направлению подготовки
09.04.01 – «Информатика и вычислительная техника»

GRADUATION THESIS (MASTER GRADUATE THESIS)
Field of Study
09.04.01 – «Computer Science»

Направленность (профиль) образовательной программы «Анализ данных и искусственный интеллект»
Area of Specialization / Academic Program Title: «Data Analysis and Artificial Intelligence»

Тема / Topic	**Простая система рекомендации корректиных действий, основанная на измерениях репозиториев программного обеспечения / A simple framework to recommend corrective actions based on the measurements of software repositories**

Работу выполнил / Thesis is executed by	**Данякин Кирилл Дмитриевич / Daniakin Kirill Dmitrievich, Жолха Фирас / Jolha Firas**	подпись / signature
Руководитель выпускной квалификационной работы / Graduation Thesis Supervisor	**Суччи Джианкарло / Giancarlo Succi**	подпись / signature

Иннополис, Innopolis, 2022

Contents

List of Tables

List of Figures

Abstract

Several works attempted to establish procedures to individuate bugs, defects, or anomalies during the different phases of software development, especially in the implementation phase. The mere detection of anomalies is not sufficient, though, at least until they get fixed. Corrective actions can be formulated to remove anomalies and enhance the software quality. To know whether an anomaly exists in a software, one must measure the software quality attributes related to it using specific software metrics. The main aim of this work is to highlight the industrial challenge in managing software development issues and find out and explain how to meaningfully attribute software metrics to useful corrective actions.

We have conducted a systematic literature review, where we have collected three kinds of data (metrics, anomalies, actions), which helped us individuate the dimensions of the problem. We found 384 software metrics, which are used to detect 374 anomalies related to 494 corrective and preventive actions.

Our findings show the need to formulate remedial strategies and build tools to automate the process of determining actions from abnormal metric values. Therefore, we propose a simple framework for detecting anomalies in software projects by using the measurements of the corresponding GitHub repositories and recommending corrective actions where needed. In this framework, we use clustering of software repositories, graph neural networks, and topic modelling.

Chapter 1

Introduction

According to the Standish Group [1], 66% of technology projects struggle or fail. Failures occur more than we expect for a number of reasons, including project complexity [2], [3], issues with methodology [4], [5], requirements in management [4], [6], time pressure [7], [8], poor communication [3], [5], key members turnover or defection [9], scarce team motivation [10]. For example, the Sydney Water Corp. lost 33.2 million dollars due to numerous change requests while automating its billing system. Analogously, London Stock Exchange lost approximately 600 million pounds due to the complex design of a new stock system that they failed to implement appropriately [3].

In many cases, however, failures could be predicted and even avoided [11], [12]. In this context, it is worth noting that many companies do not consider the task of preventing project disruptions as one of their top priorities [13], even if such failures may harm both the companies and their employees [3], [12]. This can be probably explained by a lack of strategic thinking, which invites companies to focus on generating immediate profits at the expense of more profitable and sustainable long-term returns [14].

Yet, a number of researchers in the software engineering community, intending to prevent project failures, proposed a variety of solutions aimed at improving both software development management [15] and process [16]. However, it is very difficult to choose a suitable management style or process because of the existence of a variety of factors that lead to software project failure, such as management style [4], stakeholders involvement [3], team [10], etc. Based on the critical analysis of 70 different recorded failed projects, [8] observed that for each of them, there is a wide spectrum of circumstantial factors (such as environmental and psychological ones) that could be considered the root causes of failures [10]. It, therefore, seems that there is a strong need for objective and universal assessments of software projects, which could be achieved through the usage of metrics. The possibility of providing "genuinely improved management decision support systems" based on the adoption of specific software metrics has already been established [17]–[19].

Furthermore, the ability to express subjective factors (like motivation) in numeric values increases the possibility of applying artificial intelligence techniques or methods (such as machine learning) in the field. Artificial models can effectively assess software project risks as demonstrated by [20]. A study even attempted to specify a series of metrics to describe three possible project outcomes, such as failure, smooth development, and success [21].

Other researchers, such as [22], identified several management areas where artificial intelligent technologies or techniques could be applied. These include: *a)* estimation of project success; *b)* identification of critical success factors; *c)* relatedness to project budget; *d)* connection to projects schedule; *e)* project planning; *f)* relatedness to risk identification.

Yet, there is no item on this list that suggests actions that could be taken to prevent failures. However, [23] attempted to specify the relationship between software metrics and corrective and preventive actions.

The results we reviewed above suggest that the field is ripe for further developments and perhaps even for some breakthroughs. In this work, we addressed the problem concerned with suggesting corrective actions through the detection of anomalies. Software defect prevention can be applied to any stage of software development; however, it is better to apply it in the early stages to avoid anomalies from being transferred to the next phases of the product [24]. In this respect, [25] found that a defect prevention approach grounded on inspection and testing can spot and detect between 99%-99.75% of global defects.

Corrective actions are typically defined as actions taken to eliminate the causes of non-conformities (anomalies) or other undesirable situations [26], [27]. These actions are grounded on the idea that analysing the root causes of anomalies may give us a chance to control them and, hence, to take corrective actions for avoiding their recurrence [28].

The goal of this thesis is to build a simple framework for recommending corrective actions based on the measurements collected from software repositories. The thesis is a mutual work, which consisted of the systematic literature review (SLR) that collects and categorizes actions currently applied to software development and experimental works which practically describes the implementation details of the suggested framework and resembles the results obtained from the SLR. The SLR provides us with a state-of-the-art summary of current research on the topic of corrective action recommendation and will also contribute to highlighting issues that need to be solved as well as open gaps in the literature.

This work formulates the basis for metric-based action recommendation framework. The suggested framework is simple since its diverse parts can be developed and it will provide an open research area for the researchers and practitioners. We focused the framework on GitHub repositories and used repository clustering, issue and commit analysis. This work will be beneficial for researchers, developers, and for anyone interested in software management, as it will allow them to: *a)* easily individuate actions that can be taken to prevent a software failure; *b)* identify the most promising research directions for future work; *c)* develop new, more effective strategies for dealing with managerial failures.

This work is organized as follows: Chapter 2 outlines our research protocol for the SLR, the background on the topics of this work as well as the related works; in Chapter 3, our methodology is presented, the hypotheses are explored, and the building blocks of the framework are explained. Chapter 4 covers the experiments conducted to form the components of the framework; Chapter 5 reveals the main results of our work and offers a critical interpretation of our findings and specific answers to our research questions. We also evaluated the limitations and various shortcomings potentially affecting the SLR and experiments that we performed, while Chapter 6 summarizes what we achieved and points out future research plans and directions.

Chapter 2

Literature Review

2.1 Systematic Literature Review on Metrics and Actions

2.1.1 Research Protocol Development

The Systematic Literature Review (SLR) is a sort of literature review in which several research studies or publications are collected and critically analyzed in a systematic manner [29]. We chose this type of literature review as our topic is novel and we have identified no prior literature reviews or surveys conducted that relate to this topic. SLRs are reliable, thorough, unbiased and reproducible audited tools [30] that allow the gathering, extracting, synthesising, and summarising all scholarly research on a particular topic while providing evidence or suggestions for practice, industry, and/or policy-making. This SLR, guided and inspired by [31]–[34], uses quantifiable and measurable properties of software project artifacts (software metrics) as predictors to suggest corrective and preventive actions (CAPAs) for project managers. Therefore, the key goal of this SLR is to investigate the relationship between software metrics and CAPAs, and it can be restated by utilizing a *GQM* process goal template [35].

Analyzing *software metrics*
for the purpose of *suggesting effective corrective and preventive actions*
with respect to *software engineering development stages*
from the viewpoint of *software development organizations*
in the context of *managing software development projects*

Crucially, though, no previous SLR has been found to examine the use of software metrics in such a context, attesting to the existence of an interesting gap in the literature, which we set out to explore.

In the following subsection, we present the protocol (methodology) we adopted in this work to ensure the replicability as well as the reproducibility and transparency of our findings. To do so, we followed the PRISMA checklist outlined in the Appendix A. The PRISMA checklist is a 27-item list that is meant to guide and regulate the organisation, development, and implementation of any systematic literature review.

2.1.2 Research Questions

One of the most important step in the production of an SLR involves the individuation of a research question (or of a series of research questions) that can be used to guide and inform its development. The research questions characterising this work are:

RQ1 Which software metrics are most commonly used in detecting software anomalies or triggering corrective and preventive actions?

RQ2 What are the most frequently used metrics for detecting software anomalies?

RQ3 Which software anomalies have the greatest number of action connections?

RQ4 Which software metrics are used to trigger the majority of suggested corrective and preventive actions in software development projects?

The motivation for RQ1 comes from the requirement to search, gather, and explore existing methods to measure software properties and convert them into quantifiable metrics. As a lot of papers have been published to detect anomalies or defects from software metrics, the motivation for RQ2 was to build a taxonomy of metrics used for software anomaly detection. We could effectively utilize these metrics to detect diverse anomalies that may arise at different stages of software development. The motivation for RQ3 was to better understand the relationship between the anomalies and recommended actions and to find out which anomalies triggered most actions with intention to discover the causes of such anomalies. The last question RQ4 was formulated with intention to investigate the most effective software metrics capable of initiating the greatest amount of corrective and preventive actions following the detection of anomalies.

2.1.3 Literature Search

This section explains the research approach and methodology used to gather the studies included in our final reading log.

Search Keywords. We extracted a series of relevant keywords from the four research questions formulated above, which we believed might adequately characterise them. The keywords we selected for this SLR were: *a)* "digital artifact", *b)* "metrics", *c)* "measurable properties", *d)* "software attribute measurement", *e)* "software management", *f)* "preventive and corrective actions", *g)* "software process decision support", *h)* "software defects", *i)* "software faults", *j)* "software anomalies".

Selected Databases. We selected a series of databases, which we used to perform our initial searches. The databases we selected were: *a)* ACM Digital Library, *b)* Google Scholar, *c)* Scopus, *d)* ScienceDirect, *e)* IEEE Xplore, *f)* SpringerLink.

We recognize that the list of selected databases might not be the most comprehensive; however, we also note that these databases are among those that researchers most frequently use in their daily practice. In addition, these databases gather typically scholarly research from the most relevant sources in Computer Science and Software Engineering.

Search Queries. We arranged the selected keywords into more focused search queries by using Boolean operators. The list of the search queries we developed follows below:

Query 1: ("corrective action" OR "preventive action") AND "software" AND "project" AND "management" AND ("metric" OR "measure")

Query 2: ("corrective action" OR "preventive action") AND ("software" AND ("fault" OR "anomaly" OR "defect") AND ("detection" OR "prediction"))

Table I: Preliminary results.

Searched Databases	Search Query			
	Query 1	Query 2	Query 3	Query 4
Google Scholar	17,000	16,700	4,280	4,000
ACM Digital Library	291	305	552	125
Scopus	35	72	72	957
ScienceDirect	5,350	3,993	1,565	1,387
IEEE Xplore	2	14	4	47
SpringerLink	5,524	4,205	2,699	2,531

Query 3: ("software artifact" OR "project artifact" OR "digital artifact") AND (property OR "attribute") AND ("measure" OR "metric")

Query 4: ("anomaly" OR "defect" OR "fault") AND ("software measure" OR "software metric")

We used these search queries to perform preliminary manual searches on the databases specified above. Table I displays the results of our preliminary searches.

Inclusion and Exclusion Criteria. We applied a set of inclusion and exclusion criteria to determine which papers obtained from our preliminary searches should be included in the final reading log. In other words, the inclusion and exclusion criteria were used to ensure that only the most relevant sources were accepted in the SLR. The inclusion criteria (IC) we used in this SLR were as follows:

IC1 The paper is published in a peer-reviewed journal or in a conference;

IC2 The paper is related to software engineering;

IC3 The paper uses software metrics to detect anomalies/defects/faults;

IC4 The paper uses software metrics to trigger preventive and/or corrective actions;

IC5 The paper provides quantitative measurements of software artifact properties/attributes or metrics;

IC6 The paper reports adopted preventive or corrective actions with respect to detected anomalies/defects/faults.

The exclusion criteria (EC) we used in this SLR were as follows:

EC1 The paper is not written in English;

EC2 The paper is an abstract, poster, summary, keynote, opinion piece, editorial (short paper) or it does not contain a significant amount of information;

EC3 The paper is a duplicate;

EC4 The paper can be considered as grey literature, e.g., a blog post, an unpublished manuscript or a graduate dissertation;

EC5 The paper does not provide clear descriptions of detected anomalies;

EC6 The paper deals with anomalies without suggesting preventive actions, corrective actions, or appropriate metrics for predicting them;

EC7 The paper does not satisfy IC1 or IC2;

EC8 The paper does not fulfill at least one of the criteria IC3, IC4, IC5, IC6;

Table II: Papers selection procedure.

Source	Initial selection	Removed papers									
		IC1	IC2	EC1	EC2	EC3	EC5	EC6	EC7	EC8	EC9
ACM	1273	31	472	-	5	41	73	81	-	-	69
Google Scholar	41980	7849	-	6433	8735	1481	3436	2137	8647	1464	1785
Scopus	1136	-	452	2	4	-	18	80	-	368	92
Science-Direct	12295	-	-	-	2983	137	612	807	4711	2797	231
IEEE Xplore	67	-	-	-	-	31	2	4	2	9	16
Springer-Link	14959	-	-	54	5248	306	524	469	6924	1407	17

EC9 The paper does not contain any measurable anomalies or actions (more on this in Section 2.1.4).

Results by Databases. We then applied these inclusion and exclusion criteria to the results we preliminarily obtained as listed in Table I and finalized our reading log in Table II. Figure 2.1 summarises this process graphically for the reader.

Quality Assessment. To assess the quality of the studies included, we subsequently formulated a set of questions (or quality criteria):

QA1 Does the study provide concrete examples of anomalies, metrics and actions?

QA2 Is the data collected from research conducted in industry and laboratories or just interview-based observations?

QA3 Does the study measure the impact of implemented actions?

QA4 Does the study provide quantitative measurements of the metrics?

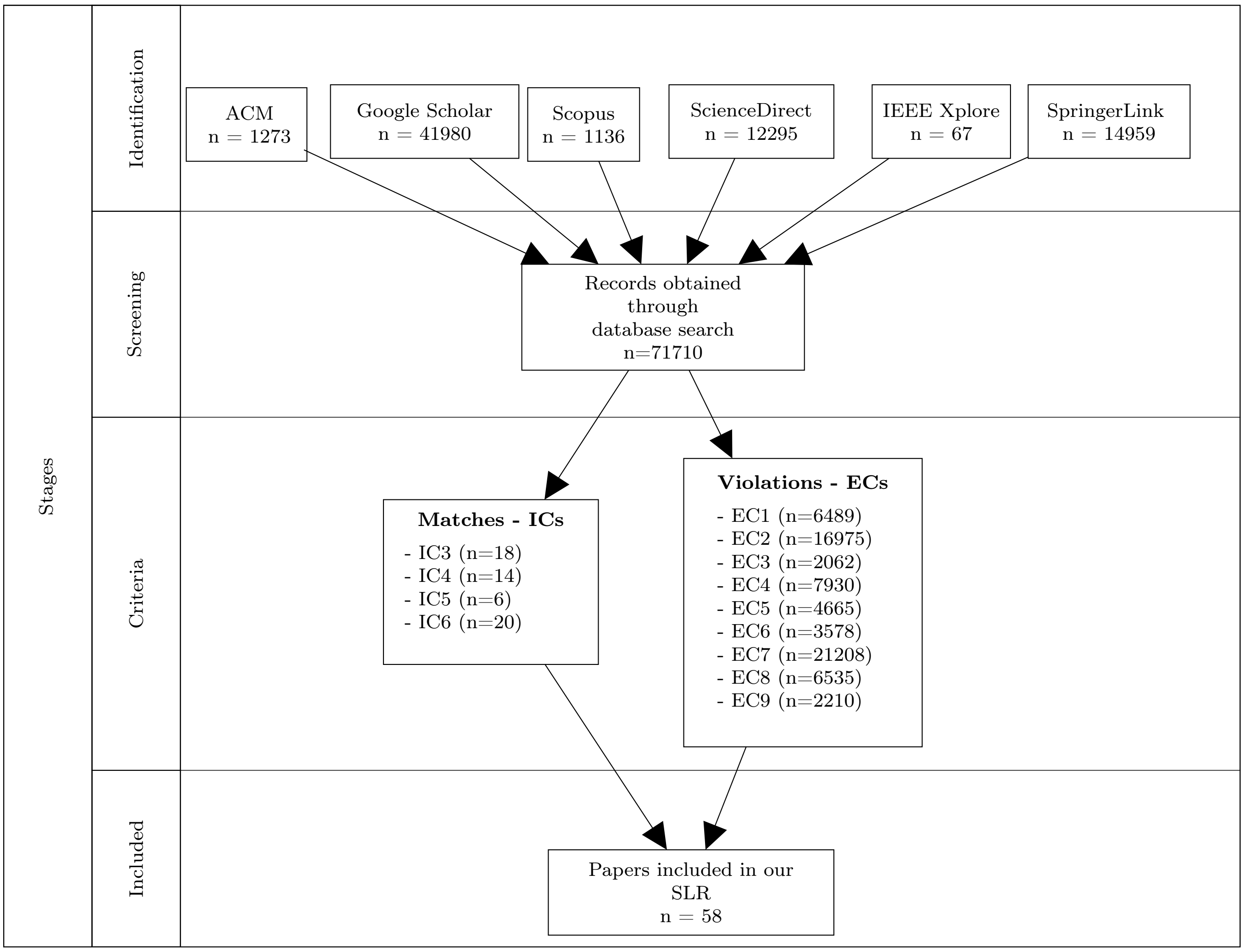


Figure 2.1: Prisma Flow Chart Diagram [36].

QA5 Are the root causes of the suggested preventive or corrective actions presented in the study?

We evaluated the quality of each article in the final reading journal by assigning a weight to each of the quality criteria listed above. Specifically, we assigned 1 if the paper fully satisfied the question, 0.5 if it partially satisfied the question, and 0 otherwise.

Criteria for QA1 were the following:

Fully matched – The study provides examples of anomalies, metrics and actions;

Partially matched – The study includes examples of any two data types (metrics, anomalies, actions);

Not matched – The study contains only examples of actions or metrics.

Criteria for QA2 were the following:

Fully matched – The data provided is industrial or experimental data;

Partially matched – The data provided data is observational data;

Not matched – No data is provided.

Criteria for QA3 were the following:

Fully matched – The study provides quantifiable impact measurements of the implemented actions;

Partially matched – The study examines the impact of some actions or presents the impact measurements graphically or textually;

Not matched – The study does not measure the impact of the implemented actions.

Criteria for QA4 were as follows:

Fully matched – The study adopts a quantitative research methodology by presenting results in the form of experiments;

Partially matched – The study adopts a qualitative research methodology by presenting results in the form of interviews;

Not matched – The study does not provide the results of any empirical research.

Criteria for QA5 were the following:

Fully matched – The paper discusses the root causes for triggering actions;

Partially matched – The study provides root causes for some actions or only the cause category;

Not matched – The study does not mention the root causes for triggering actions.

We then ranked the quality of the papers included in the reading log. The results of this ranking are shown in Appendix B.

We noticed from Table III that the vast majority of the papers we included in our final reading log ($\sim 88\%$) were of good or excellent quality. Hence, we can infer that the data on which we based our SLR are scientifically sound. We present the results and discussion sections of the SLR in Chapter 5.

Table III: Papers' Ranking.

Quality Level	Score Range	Count	Percentage
Poor	[0-1.5]	7	12%
Good	[2-4]	43	74%
Excellent	[4.5-5]	8	14%
Total	—	58	100%

2.1.4 Results

This section summaries the results obtained from reading and extracting the data from the papers we included in the final reading log.

Preliminary Clustering

As noted above, we performed this SLR on some of the most popular and trusted databases available to researchers. Figure 2.2 describes the distribution of the papers included in our final reading log by databases.

We attributed papers to single databases (even if some papers could be found across different databases), based on the chronological order of the searches performed. We also noticed that two databases (Google Scholar and ScienceDirect) indexed 52% of the papers included in our reading log. Searches performed on IEEE Xplore and Scopus rendered the least number of relevant papers amounting to only 12% of the total papers included in the log. We further notice that we searched IEEE Xplore and Scopus by using only papers' metadata.

We also present in Figure 2.3 ("#" stands for the "number of") the geographical distribution of the studies we included in our log (based on first author's affiliation). Figure 2.3 shows that the studies we selected have been

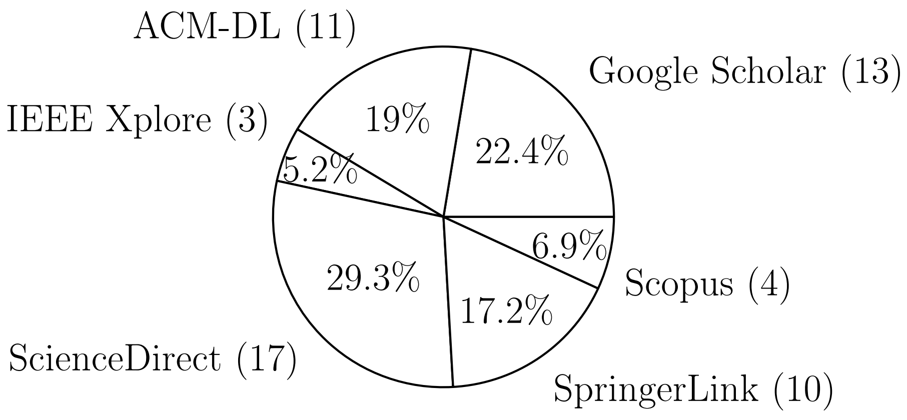


Figure 2.2: Distribution of studies by database.

published in 18 countries. This ensures a sufficient degree of cultural diversity for our reading log.

Studies Classification

For this SLR, we decided not to select a year as the starting point for our searches, as we wanted to review the entire field comprehensively. This is why our reading log consisted of papers published both in the 20th and in the 21st century. Figure 2.4 shows the distribution of the papers we included in our reading log by year of publication. While the first paper we found on the topic dates back to 1981 [37], about 57% of the studies selected were published in the last 10 years. We further classified the studies by publication type and figured out that 39 papers, which is 67% of the total included in our log, were published in peer-reviewed journals, while the rest 19 (amounting to 33% of the total) came from conference proceedings.

We then extracted more significant data (such as metrics, anomalies, actions) from the papers we included in the log; hence, we clustered them to gauge their relevance and significance for our research objectives. For this reason, we extracted several attributes (see Appendix D) for each of the three

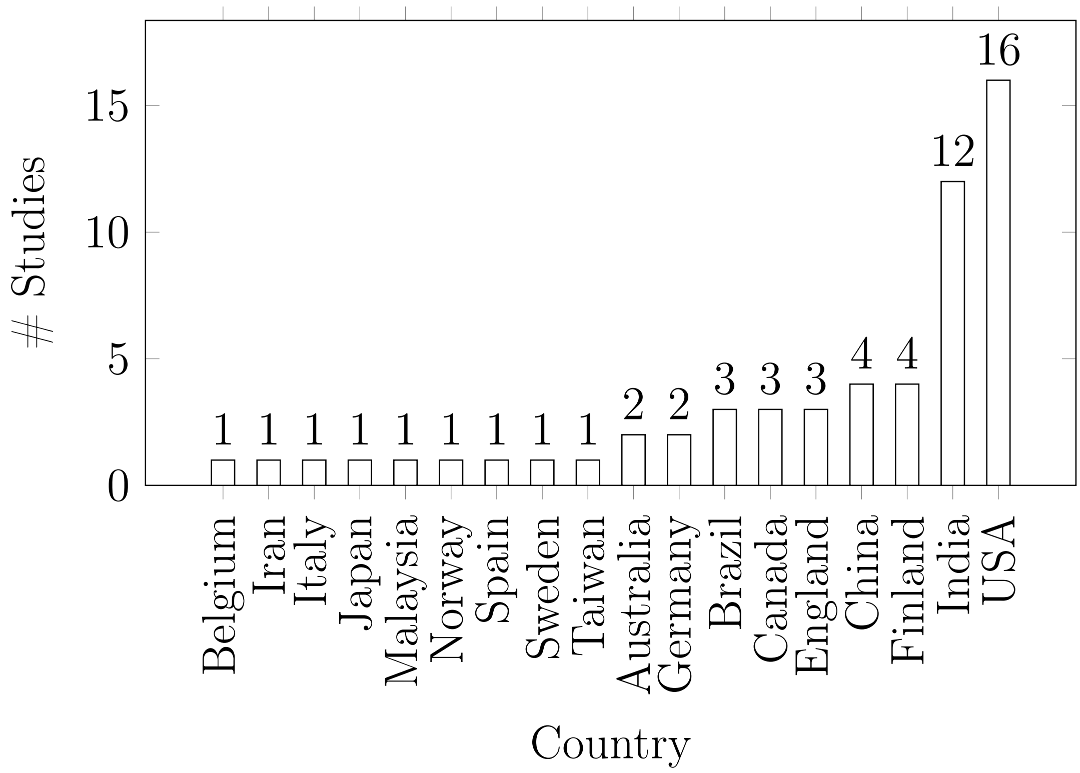


Figure 2.3: Distribution of studies by country of origin, based on first author's affiliation.

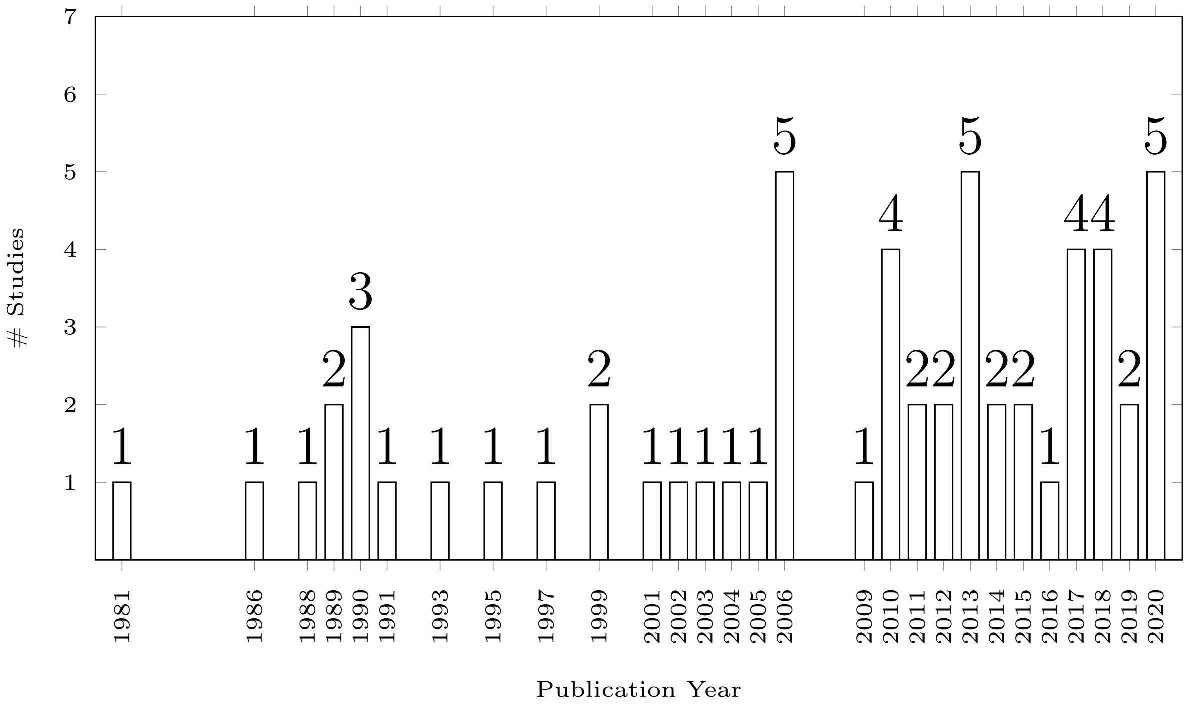


Figure 2.4: Distribution of studies by year of publication.

data types we wanted to investigate. This was not an easy task, as most of the data we gathered was heterogeneous.

Metrics Data

This subsection presents the findings related to the existing software metrics we collected from the papers we analysed. 43 studies out of 58 provided this kind of data. The study [38] provides a taxonomy of software metrics (~ 500) used for software fault detection. Since it basically covers most metrics data collected in this research, we considered it as an outlier regarding the other included studies, and our analytical results in Section 2.1.4 do not take into account this study. Table IV shows the metrics mostly used in the studies we analyzed, so it does not present all the metrics we collected. Other metrics were used either in two studies or in a single study. Among the 42 relevant studies we reviewed, we found 12 studies that used "Lines of Code" as a metric for multiple purposes. [39], [40] used it for faulty classes prediction, [41] used it for predicting software reliability, [42] used it for monitoring project progress, [43] used it for software quality control, [44] used it for identification of high-risk components in software development process, [45] used it for action-oriented process improvement, [46] used it for corrective action identification for process improvement, [47] used it for evaluating the maintainability of model-to-model transformations, [48] used it to early detect the vulnerable files in software system, [49] used it for predicting unusual events in GitHub repositories, [50] used it for defect prediction in finance software system.

The study [51] classified software metrics into three dominant classes: product, process and resources. Inspired by this approach, we also classified the metrics found in the papers we reviewed. Most metrics belonged to the

Table IV: Metrics distribution in the final log.

Metric	Count	Used in Studies
Lines of Code	12	[39]–[50]
Cyclomatic Complexity	7	[39], [40], [42]–[44], [48], [52]
Number of Change Requests	6	[42], [43], [49], [52]–[54]
Coupling Between Object Classes	4	[39], [48], [55], [56]
Defect Density	4	[52], [57]–[59]
Documentation Size	4	[42], [43], [54], [60]
Number of Defects	3	[57], [61], [62]
Fan-in/Fan-out	3	[42], [48], [56]
Productivity in function points	3	[45], [63], [64]
Response For a Class	3	[39], [48], [55]

"process" category. Specifically, we observed the following distribution: "product" (125 references, 33%), "process" (260 references, 68%) and "resources" (4 references). The above shows that the first two categories ("product" and "process") amounted to 99% of the total, while the latter category ("resources") accounted for 1% of the total.

Furthermore, to better understand the usage of metrics in the studies we selected, we clustered them into two major categories: i) metrics used as anomaly predictors, and ii) metrics used as action triggers. The first category (Table V) covers metrics used for predicting software anomalies and handling them by taking relevant corrective and preventive actions; whereas the second category (Table VI) describes the metrics' usage for recommending the most useful actions in order to correct or prevent an anomaly not explicitly specified. In this research, we defined an "anomaly" or better a *"measurable anomaly"* as the deviation of software metric values from the expected measurements that triggers an appropriate action(s) for fixing this anomaly [65]. We also defined a *"measurable action"* as the action that is triggered by abnormal metric values

Table V: Top software metrics ranked by number of anomalies detected.

Metric	Detected Anomaly Count	Used in Studies
Defect injection metric	30	[66]
Defect density	10	[57], [58]
Frequency of interactions / number of meetings	10	[54]
Lines of Code	7	[39], [40], [44], [45], [48], [49]
Cyclomatic Complexity	4	[39], [40], [44], [48]
Number of Change Requests	3	[49], [53], [54]
Number of Defects	3	[57], [61], [62]

Figure 2.5: Distribution of software metrics used as anomaly indicators.

and implemented to recover the expected state of the software development project.

We reviewed 27 studies that used software metrics as anomaly predictors. We found 155 metrics used for detecting 124 anomalies. The distribution of the most frequently used metrics for software anomaly detection is shown in Section 2.1.4. "Lines of code" was the most commonly used metric. Only 4 metrics are presented in the figure; this is because the other metrics (151) were used either in two studies or in a single study.

We further note that 125 metrics were used as action triggers without identifying the specific anomaly encountered; yet, some papers reported the root causes of the actions taken. We remark that these 125 metrics were gathered from 19 studies and were used to recommend 141 actions (31 corrective, 54 preventive and 56 enhancement actions). Almost most of these metrics (109 out of 125 metrics) are process metrics and the distribution is presented

Table VI: Top software metrics ranked by number of triggered preventive and corrective actions.

Metric	Triggered Action Count	Action Category	Used in Studies
Defect injection metric	30	Preventive	[66]
Defect density	19	Preventive	[58]
	4	Corrective	[57]
Frequency of interactions / number of meetings	10	Corrective	[54]
Number of Defects	3	Corrective	[57], [61], [62]
Number of Change Requests	2	Corrective	[53], [54]

in Figure 2.6. The metric "number of defects" was the most frequently used (in 3 studies), namely in [57], [61], [62]. Figure 2.6 does not present the distribution of the remaining metrics (122) - those that were used uniquely as action triggers. Also, only the preventive and corrective actions are considered here, since enhancement actions are not employed to handle anomalies. In addition, the metric "productivity in function points" was the most frequently used metric in 2 studies [45], [59] for triggering 14 enhancement actions.

Anomaly Data

The second type of data extracted from the papers we included in the reading log concerned anomaly data. According to IEEE 1044-2009 standard [65], an anomaly is defined as any abnormality from expectations and it is used as a synonym for error, fault, failure, incident, flaw, problem, gripe, glitch, defect, or bug. Indeed, we extended this definition to become the "measurable anomaly" as stated in Section 2.1.4 by considering the software metrics as the

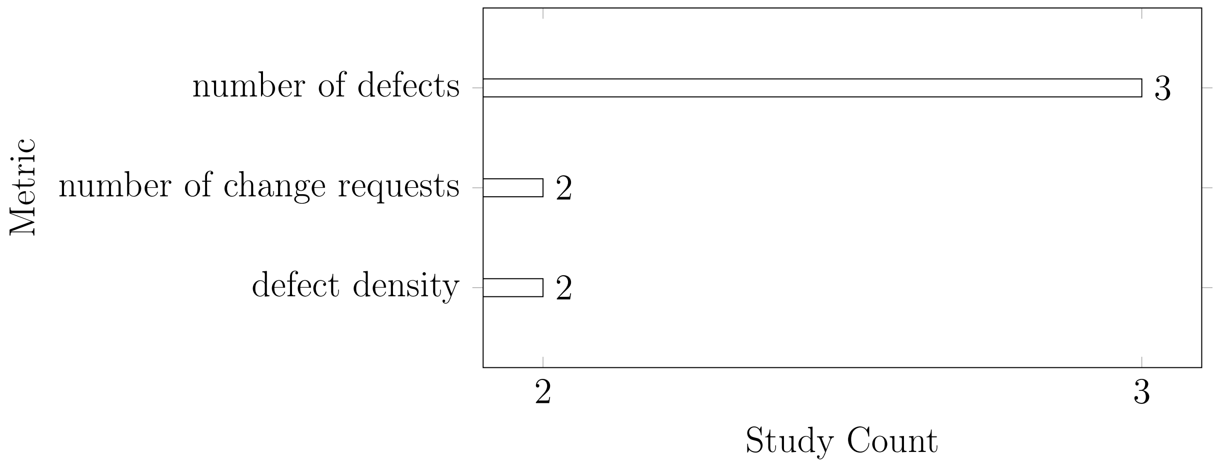


Figure 2.6: Distribution of software metrics used as triggers of corrective and preventive actions.

primary indicators for these anomalies. We managed to extract 374 anomalies from 45 studies with 142 specified root causes. A Root Cause is described as the fundamental cause of any problem that, when identified and treated, can result in the resolution of the problem [67], [68]. Root Cause Analysis (RCA) is a technique for identifying the causes of all deviations throughout the project's life cycle and is used to both remedy and avoid deviations by implementing corrective and preventive actions [69]. Some studies used this technique to find and report root causes for the anomalies encountered, as in [63], [69], [70].

Table VII shows the top preventive and corrective actions used in the studies we selected. The figure does not include enhancement actions, as these actions were typically used for increasing the quality of the software process and/or product.

Actions Data

In this subsection, we present the findings related to the collected actions and their relationships with other data types. We gathered 494 actions, which we categorized into three main classes (corrective, preventive and enhance-

Table VII: Top software anomalies ranked by the number of suggested actions.

Anomaly	Action Category	Action Count	Detected in Studies
Class resistance to change	Preventive	27	[71]
	Corrective	2	
Inaccurate estimation of efforts	Corrective	15	[72]
Systemic defects into software	Corrective	1	[73]
	Preventive	11	
A lot of bugs encountered	Corrective	10	[72]
Insufficient outcome of development work	Corrective	10	[74]
Oversight issues	Preventive	8	[75]
Inadequately performed technical activities	Corrective	7	[42]
Casual defects of operations	Preventive	5	[76]
Lack of information on how the system should work	Corrective	5	[72]

Table VIII: Top software anomalies ranked by number of metrics used in detection.

Anomaly	Used Metrics Count	Detected in Studies
CI build failure	33	[77]
Faulty classes	18	[39]
Vulnerability files in software	13	[48]
Unusual pull request	10	[49]
Unusual commit	9	[49]
Fault-prone classes	6	[40]
Unusual GitHub issue	6	[49]
High risk components in software	4	[44]
A cyclic group of components in OO software	3	[55]
Functional defect in the software system	3	[60]
Nonfunctional areas of design not adequately addressed	3	[61]
Incomplete/Inaccurate documentation of software modules	3	[54]

Table IX: Top preventive and corrective actions ranked by the number of metrics used in triggering the action.

Action	Used Metrics Count	Reported in Studies
train developers to understand the requirement errors	6	[20]
emphasize nonfunctional areas in succeeding phases	3	[61]
make requirement analysis and fix the cause	3	[78]
reduce or avoid cyclic dependencies in the development stage	3	[55]
team members should make sure that knowledge is appropriately distributed amongst them. For example, pair programming, code review are practices which can promote knowledge sharing.	3	[54]

ment). This categorization is inspired by [64]. Corrective actions represent the actions applied to fix or correct the anomaly, while preventive actions are used to avoid the recurrence of the anomalies in the future. Enhancement actions are actions adopted to increase software quality. Our results can be summarised as follows: out of 494 possible actions, we gathered 250 preventive, 176 corrective, and 68 enhancement actions. We understand from the results displayed in Figure 2.7 that more than half of the actions are categorized as preventive actions. This is very reasonable because the longer an anomaly remains in the production line, the more costly it is to correct or fix it [79]. The goal of anomaly prevention is to discover those anomalies early in the product's life cycle and hence to prevent them from spreading or reoccurring. Therefore, it is necessary to implement actions to prevent anomalies from being introduced into the product in the early phases of a project development [58].

As we stated above, actions are employed to improve the software pro-

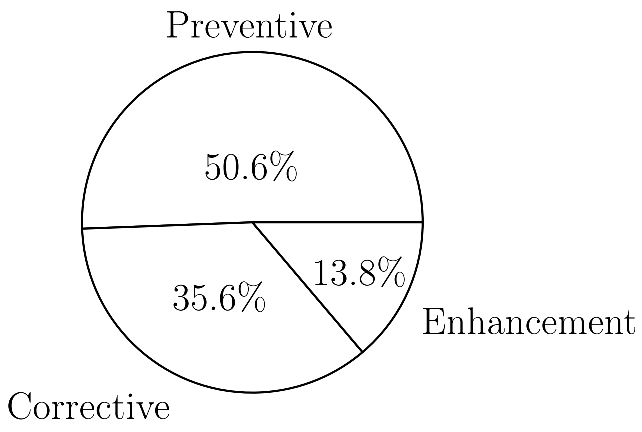


Figure 2.7: Distribution of action categories.

cess and enhance the product's quality [58], [71], [80]. Capability Maturity Model® Integration (CMMI®) for Development is a collection of best practices for product and service development and maintenance [81]. It covers processes that span the whole lifespan of a product, from creation through delivery and maintenance. We mapped out our results into one of the most widely used model for software process improvement, so we classified the actions collected from the studies we analyzed into CMMI key process areas. This allowed us to more clearly see the relevance and significance of our findings to software process improvement and to point out a gap in the existing literature regarding action taxonomies, as shown in Section 5.1.4.

Our results, presented in Figure 2.8, show that most of actions belonged to either Integrated Project Management (IPM) or Technical Solution (TS) process areas. It is worth noting that we did not find actions for all process areas and that the distribution of actions we report was unequal. These findings call for the need for more works on minimally studied process areas, as they may also trigger and generate a lot of anomalies. The reader can find the descriptions, as well as the goals of CMMI key process areas in [82].

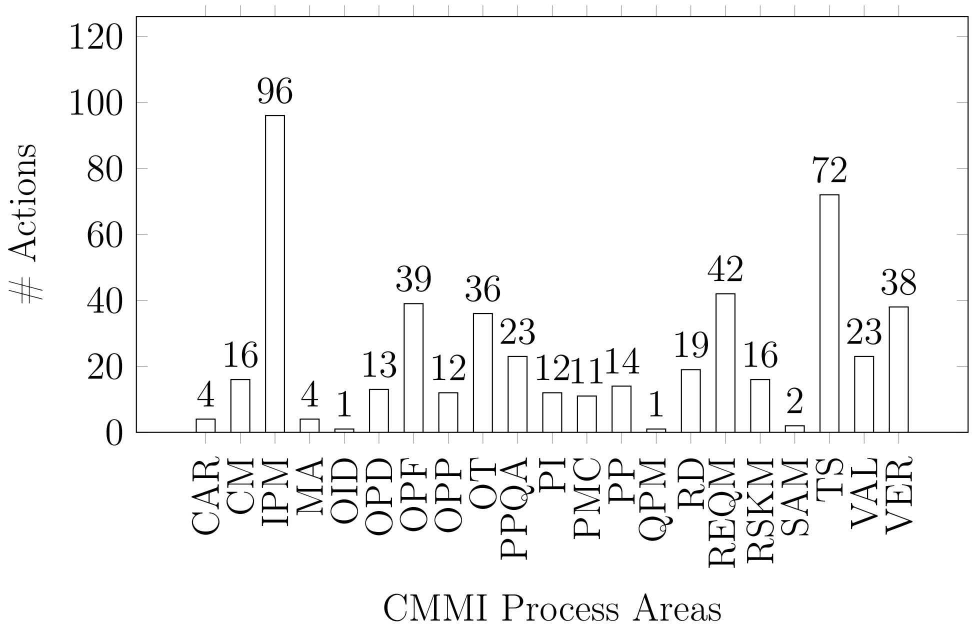


Figure 2.8: Distribution of actions by CMMI key process areas.

2.2 Background on Clustering Software Repositories

There are a vast number of aspects involved into the development of a software system related to code, people, and processes. Software repositories such as those stored on GitHub contain the information about these aspects. Therefore, a retrospective analysis of such software repositories can provide valuable insights into the evolution, growth, qualitative characteristics, and problems of the corresponding software development projects. The insights gained through such retrospective analysis can affect the decision-making process in a project, and improve the quality of the software system being developed [83].

To conduct such retrospective analysis, Munaiah et al. (2017) [83] proposed to classify software repositories as "engineered" or "not engineered", i.e., they proposed to group the repositories based on the similarity of their attributes. However, such software ecosystems as GitHub contain more than 290 million repositories and more than 87 million users, and manually labeling them is a time-consuming process as shown by Borges and Valente (2018) [84] when they focus only on 5000 repositories due to the fact that they had to label the repositories manually. For this reason, many studies use unsupervised learning techniques such as clustering to determine the similarity between software repositories and to provide ground for repository analysis.

Determining software repository similarity is an essential building block in studying the dynamics and the evolution of such software ecosystems as GitHub [85]. For instance, Borges and Valente (2018) [84] determined similar repositories with clustering methods to understand the growth patterns of GitHub repositories' number of stars, which leads to better understanding of how people star the repositories and what this starring is attributed to.

Even though our work is aimed at clustering of software repositories based on the corresponding metrics data, we briefly review the proposals to classify repositories manually and point at their shortcomings to see if the clustering used in related works corresponds to the proposed manual classifications and whether it addresses the shortcomings of manual approaches. We also note which of the manual classifications our study is related to the most. Only then we review the work on clustering software repositories and describe their limitations.

2.2.1 Existing Proposals to Manually Classify Software Repositories

Munaiah et al. (2017) [83] classified repositories as "engineered" or "not-engineered" with the intention of identifying high-quality GitHub repositories and removing noise (e.g., repositories for homework). “Engineered” is a repository that provides general-purpose utilities to users other than the owners and is similar to those of Amazon, Apache, Microsoft, and Mozilla. They manually classified 450 GitHub repositories based on this definition and two of its aspects: “organization” and “utility”. The main limitation of this approach is a bias towards industry giants and their repositories under the assumption that they use the “sound software engineering practices”.

Borges and Valente (2018) [84] performed the large-scale classification of application domains on GitHub based on the top 5000 public repositories by the number of stars. The top-3 domains by the number of projects were web libraries and frameworks, non-web libraries and frameworks, and software tools – they can be viewed as meta-projects used to implement other projects. Borges and Valente (2018) [84] hypothesized that the application domain is useful for understanding the clusters of GitHub repositories based on number of stars history over time. However, they noticed that there is no statistical difference between the number of stars of systems software, applications, web libraries and frameworks, and documentation, so under the perspective of clustering using number of stars evolution, these domains can be grouped into one, resulting into three-domain repository classification.

Altogether, the limitation of the domain-based classification is that several domains can be grouped together into one or divided into a number of smaller domains, therefore, such grouping is subjective and it is hard to precisely determine the number of separate domains in advance.

Treude et al. (2018) [49] aimed at manually identifying “unusual events” in 200 randomly sampled GitHub projects with at least 500 commits and at least 100 pull requests or 100 issues based on projects’ repository metrics. Such unusual events and the association between events and metrics were determined via a survey of 140 software developers responsible for or affected by these unusual events. The repositories were classified as having unusually large or small values (based on the "extreme outlier" definition) in these types of events:

- Commit-related events – the most useful metrics to detect these events according to the survey with developers are "the number of lines of code modified/deleted/added in a commit";
- Issue-related events – with useful metrics for detection being "days between open and closed" and "number of comments for label";
- Pull request-related events – detected by "number of comments".

Treude et al. (2018) [49] claimed that this classification can "help prevent potential problems early on, encourage discussion where it is needed, and give important pointers to events in a project's history to be reviewed". According to the findings of Treude et al. (2018) [49] , the limitation is that awareness tools based on commit or source code activity alone are not sufficient to communicate all the information developers care about in a project, which means that additional data related to issues and pull-requests have to be collected and analyzed.

Out of the proposed approaches to classify software repositories, our study corresponds more to the one by Treude et al. (2018) [49] , as we try to identify the metrics that would characterize software repositories qualitatively and signal about potential anomalies. To determine a set of metrics that are useful for anomaly detection, Treude et al. (2018) [49] rely on the opinion of software developers who caused or deal with these anomalies, however, they found out that developers "value simple and easily understandable metrics over complex ones", though simple and easily understandable metrics might not be enough to uncover the underlying data patterns. To address this, we conduct an automatic clustering of the repositories based on a relatively large set of GitHub metrics instead of relying on the simple "extreme outlier" definition of anomaly, and then try to identify useful metrics that indicate something unusual in the resulting clusters of repositories.

2.2.2 Clustering Software Repositories

There has been active research on clustering software repositories and corresponding software projects even if such clustering has not been always connected to the classifications mentioned in Section 2.1. We review the most relevant work in the field and present our review in the following paragraphs.

Kawaguchi et al. (2006) [86] proposed a tool that automatically categorizes software systems for identifying similar projects to facilitate reuse and sharing knowledge among software projects. As metrics for clustering, they used identifiers uncovered by latent semantic analysis from source code, and then they applied cosine similarity and unifiable cluster map to automatically categorize software repositories. Applied on 41 programs in C in five categories coming from SourceForge, their method generated 40 clusters with mostly 2-3 software systems per cluster. The limitation of such clustering is that it produces too element-specific, not descriptive, and small clusters.

Jureczko and Madeyski (2010) [87] used such clustering algorithms as hierarchical, K-means, and Kohonen's neural network to identify groups of software projects with similar characteristic from the defect prediction point of view. They measured Chidamber and Kemerer, Bansiy and Davis metrics suites, lack of cohesion in methods (LCOM3), and McCabe's cyclomatic complexity on 92 versions of 38 proprietary, open-source and academic projects, and used these metrics for clustering. The existence of two clusters was proven with statistical testing: custom-built solutions and text processing projects by medium sized international team. The limitation of the proposed clustering method is that it requires the information about defects to be predicted, therefore, it is hard to externally validate it.

Borges and Valente (2018) [84] studied GitHub repository starring practices and meaning, characteristics, and dynamic growth of GitHub stars. They used K-Spectral Centroid algorithm on top 5000 public GitHub repositories by the number of stars in order to cluster the repositories and analyze the stars growth patterns. They identified three clusters that suggest a linear growth at different speeds and one cluster with a sudden growth in the number of stars. This classification study has a limited external validity of the method as authors focus only on the characteristics of the most starred GitHub projects.

Saied et al. (2018) [88] used DBSCAN-based hierarchical clustering on 6638 libraries and 38000 client systems hosted in repositories from GitHub to group the libraries that are most frequently co-used together by clients in order to relieve developers from manual analysis. To cluster libraries, they applied usage vector for each library indicating which client systems use the library. The obtained usage patterns exhibited high usage cohesion with an average of 77%. The study has the limitation in the method's external validation – due to the data collection used, there could be duplicates and missed dependencies, which might mislead the clustering algorithm and provide biased results.

Pickerill et al. (2020) [89] clustered 1,786,601 GitHub repositories into "engineered" and "not engineered", following the classification by Munaiah et al. (2017) [83]. They extracted information about repositories' Git commit, integration, committer, integrator, and merge frequency, and, using these metrics, applied K-means clustering algorithm, which resulted in 38% of the repositories being labeled as “well-engineered”. The main limitation of such clustering is limited external validity, since the authors validated the clustering only on a small manually labeled subset of the 1,786,601 repositories, and, as they noted, there is a possibility that both the ground truth and clustering are wrong, and their agreement is coincidental.

Rokon et al. (2021) [85] claimed that determining repository similarity is an essential building block in studying the dynamics and the evolution of such software ecosystems as GitHub, and for this purpose they proposed an embedding of repository metadata, code, and structure, which they then used to cluster 1013 GitHub repositories with hierarchical clustering. This resulted into three clusters of benign, malware, and REST API related repositories, or 26 sub clusters of the three. This clustering has limited internal validity as authors used only one cluster validity index.

Tsoukalas et al. (2021) [90] divided 27 software projects from the technical debt dataset [91] into six clusters of similar projects with respect to their technical debt aspects using K-means algorithm and built specific technical debt forecasting models for each cluster using regression algorithms. As metrics for clustering, they used effort in minutes to fix code smells, bugs, vulnerability issues and number of lines of code, bugs, smells, as well as cyclomatic complexity. The results showed that the prediction errors tend to be statistically significantly lower in within-cluster technical debt forecasting than in cross-cluster forecasting. The limitation of this study is that the authors did not provide the interpretation for the resulting six clusters, which limits the external validity of the clusters.

Xu et al. (2021) [92] performed a large-scale empirical study with 40 clustering algorithms on 27 versions of 14 open-source projects to explore the impacts of clustering-based models on defect prediction performance. They used code complexity, process, and network metrics and compared the clustering-based models to supervised defect prediction models. The results showed that not all clustering models are worse than supervised models, however, authors did not provide an analysis or interpretation of the resulting clusters, which limits the external validity of such clustering.

Tan et al. (2022) [93] focused on hierarchical clustering and Bunch clustering algorithms for remodularization and provided information about their suitability according to the features of the software repositories such as bugs, code smells, duplications, number of lines of code, size, and number of stars. The resulting clusters were described in terms of how well the proposed clustering algorithms performed according to the MoJoFM metrics, however no cluster interpretation was provided. For validation, authors took top 30 GitHub repositories by the number of stars written in Java with at least 10 commits, which limited external validity of the resulting clusters.

2.3 Background on Graph Neural Networks

There has been a notable increase in the reuse of deep-learning architectures, specifically convolution layers [94]. By aggregating and repeatedly updating the information surrounding a node, the GraphSAGE model may express a goal in an inductive approach [95]. Due to the issue of over-smoothing in graph neural networks and the need to keep processing times low, this method is typically limited to one or two layers. In contrast, GraphSAGE provides inductive learning of weights that could be reused for new nodes. Graph attention network (GAT) is a graph neural network (GNN) model that employs a self-attention mechanism and linear projections, and it was developed due to the fact that not all neighbors have the same impact on an entity [96]. According to the theory, the prior model's sequence of operations has been adjusted to answer concerns that GAT's attention is static and incapable of processing the nodes' genuine influence [97]. HinSAGE, a variant of [95] implemented in StellarGraph, handles the heterogeneity of the graph, which is not addressed by the aforementioned methods. Based on the type of neighbor or relationship, HinSAGE groups and averages the neighborhood's vectors. Using this method, the attention coefficients and messages of neighbors are integrated into a single vector. Each operation employs linear projections that are class and relation-specific. This makes it challenging to deploy in the context of a knowledge graph due to the large number of distinct edges.

2.4 Background on Topic Modeling

Topic modelling finds topics in a given text corpus automatically without requiring taxonomies, tags, or training [98]. In order o build a model of related words, it utilizes frequency and co-occurrence of words in the documents belonging to a corpus [99]. Topic modelling has been applied to software engineering research in such topics as test case prioritization [100], understanding of subjects of discussion among mobile developers [101], duplicate detection in bug reports [102]. A three-level hierarchical Bayesian model, Latent Dirichlet Allocation (LDA), is the most common technique used to create topic model. In LDA, each item of a collection is modelled as a finite mixture over an underlying set of topics [99]. Topic distribution of a document is randomly sampled from a Dirichlet distribution with hyperparameter α, and each topic's word distribution is randomly sampled from a Dirichlet distribution with hyperparameter β. The former corresponds to the document-topic density – the higher α is, the more topics the document contain. The latter shows the topic-word density – the higher β is, the more words the topics in the corpus contain [103]. There is another parameter k that stands for the number of topics, and it is as well required to create a topic model using LDA. Many studies use the default settings for these parameters ($\alpha = 1.0, \beta = 0.01, k = 100$; other sources suggest $\alpha = 50/k$ and $\beta = 0.1$ [104]). Researchers have found recently that the default values do not provide to the best model fit, and for this reason, they have considered using optimization to determine optimal parameter values (e.g., [105]). To measure model fit, researchers have employed perplexity, the geometric mean of the inverse marginal probability of each word in a held-out set of documents [106], [107]. Low perplexity means the language model correctly guesses unseen words in test data.

Chapter 3

Methodology

In this chapter we describe our proposed methodology to build a framework of automatically suggesting corrective actions in software repositories. The industrial challenge that we dealt with concerns automating the procedure of raising issues based on the measures of software artifacts in a version control systems (e.g., GitHub). Raising an issue in a repository is not sufficient since this solution has been studied by a lot of researchers along the past several years. To bring a novelty in our approach, we have decided that a corrective action must be recommended to eliminate the issue as a part of the proposed framework. We propose a simple framework for this challenge as a basis for future research works.

First, we collect the measurements of software artifacts from GitHub repositories, cluster them, and conduct the software metrics analysis in order to explore the repository software metrics data, analyze software repositories, and select the metrics to use in the following analysis. Since we do not have ground truth labels available to discriminate the repositories, we follow this unsupervised learning approach.

After acquiring the “relevant” metrics and repository clusters from the abovementioned data exploration, we build the framework consisting of the following:

1) *Repository analysis*, where we relate a repository to one of the clusters, and if the repository relates to an anomalous cluster, we mark it as potentially anomalous;

2) *Issue analysis* that includes modeling the issues to specific topics based only on the title and body of the issue then predicting the links between the repository measurements and the raised issues in the repository;

3) *Commit analysis* as a follow-up step for the previous one in which we run topic modeling on textual data of fixing commits then we employ supervised learning approaches to predict the links between the raised issues and the corrective commits.

The details of the proposed approach for the software repository clustering, analysis and for building the action recommendation framework are presented in the following sections.

3.1 Clustering Software Repositories

Considering ensemble methods in machine learning, it is of interest to consider here a suite of different clustering methods providing different partitions of data and aggregate their results into a single consolidated clustering without accessing the features or methods that determined these partitions, i.e., create a consensus clustering. The solution to this problem was introduced by Strehl and Ghosh (2002) [108]. However, to our knowledge, there is no prior study applying the consensus clustering approach to analyze software repositories. We also noted that the application of consensus clustering had received comparatively little attention in the case of software metrics.

In this work, we aim to overcome the limitations (such as limited validity) of using a single clustering method with our main objectives along with key aspects of originality being the following:

- Thoughtful use of several clustering methods and building consensus results following consensus clustering as proposed by Strehl and Ghosh (2002) [108]. We discuss it as one of the key points in this section.
- A careful analysis of clustering results with the use of multiple cluster validity indices.
- Identification of relevant attributes (features) through statistical testing.

Our hypothesis is that it is possible to qualify a software repository by using its quantitative metrics. Our methodology for identifying clusters of repositories consists of three steps. The first step is data collection and preparation. We find N repositories in GitHub, using GitHub Repo Search API with approximately the same size and popularity, which are determined via repository size in kilobytes, number of stars and number of forks correspondingly. We separate these N repositories into $P > 1$ sets of $\frac{N}{P}$ repositories. In addition, we predefine the number of consensus clusters c.

The next step is processing of the repository sets. We pick the first $\frac{N}{P}$ repositories, collect n metrics from them and apply the following algorithm:

1. We normalize each metric to the [0,1] range.

2. We group repositories into c clusters based on the n metrics using m clustering algorithms with different values of parameters that we tune.

3. For each grouping resulting from the previous step, we calculate three cluster validity indices and obtain at most three best groupings (in the sense that each groping optimizes a certain validity index) for each algorithm. Refer to Section 3.4.1 for details.

4. We use the consensus clustering approach in order to build a new "winner" grouping from the best ones (refer to Section 3.4.2 for details).

5. For each group in the winner grouping, we calculate prototype vector of n metrics that is an average vector in the group.

6. We use statistical testing to mark "insignificant" metrics in the set of prototype vectors: these metrics do not change their values statistically significantly across the c clusters. Refer to Section 3.4.3 for details.

We repeat the entire algorithm for the next $\frac{N}{P}$ repositories, and so on. As a result, we get a collection of c sets of prototype vectors. After that, we identify the metrics that were marked as insignificant for all P sets of prototype vectors.

The final step of our methodology is aggregation of the results. We match the calculated cluster prototypes to compare them and the clusters they represent across P sets of clustered repositories. For that, we consider the minimum-weight matching problem in P-partite graph across all $c \cdot P$ prototypes (for details refer to Section 3.4.4), and, as a result, get c sets of P matched prototypes. We calculate the discrepancy in every such set of prototype vectors and, using a similarity measure, calculate the discrepancy value d for each of the c

sets, which shows us how different the cluster prototypes are across the P sets of data for each of the c clusters.

3.2 Repository Issue Analysis

Creating or raising issues by human reporters in software repositories is one of the ways to notify the developers about an issue or abnormality in the repository. Issue tracking tools are commonly used by software teams to manage and track the issues such as Jira, Teams, etc. However, most issue management tools involves human intervention in raising and also fixing the issues. In this thesis, we proposed an automated way to raise issues and suggest the fixing actions. In fact, GitHub issues are labeled by contributors with meaningful labels and fixed according to their priorities. Not all issues in software repositories indicate some abnormality or unusual nature in the repository. According to the recent studies on this topic [109]–[112], the researchers classify the issues into three main categories (bug report, feature request, question/query). We are interested in this thesis with “bug reports” which are raised due to some abnormality or unusalness in the software repository.

We use the Github API to extract all issues in the N repositories, then we perform two tasks on them. The first task is topic modeling on the raised issues based on their textual content in order to cluster the similar issues into the same topic which will be labeled later. The second task is prediction of the links between issue topics and the repository metrics. The output of the topic modeling task will be fed as an input to the second task as it is visualized in figure 3.1. We follow the same methodology for the commits but with a slightly different settings.

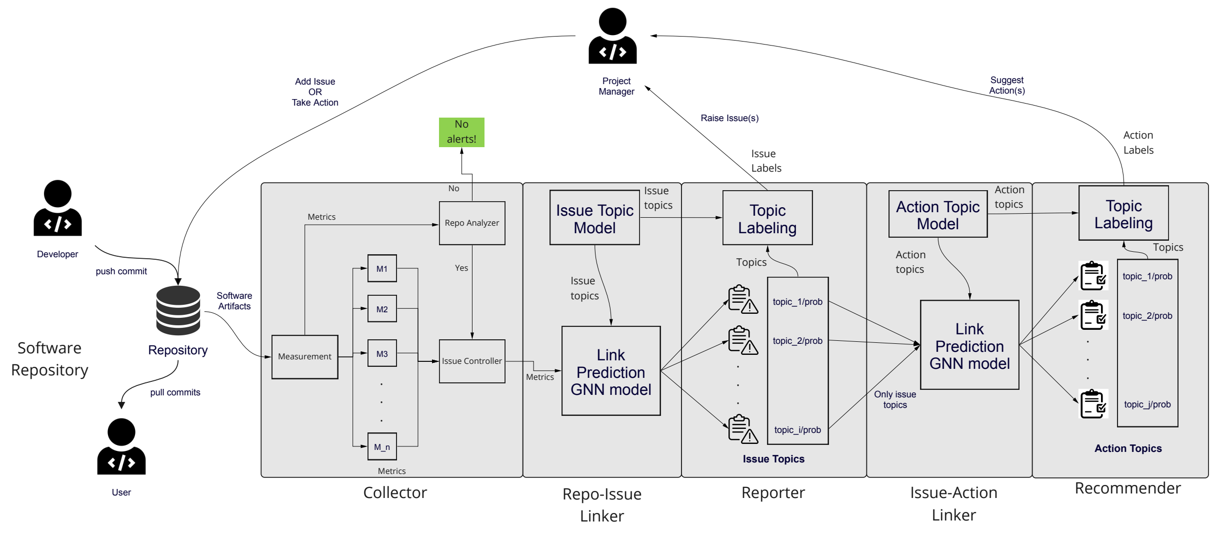


Figure 3.1: A simple framework for corrective action recommendation in software repositories.

3.3 Proposed Framework

In this section we describe the suggested framework for recommending corrective actions in software repositories. The framework mainly consists of five components as shown in Figure 3.1. The components are as follows:

1. Collector;
2. Repo-Issue Linker;
3. Reporter;
4. Issue-Action Linker;
5. Recommender.

The framework/system activates by default when a developer makes a new commit to the repository. The *Collector* measures the attributes of software artifacts in the repository and runs the clustering based anomaly detection sub-component *Repo Analyzer*. If the repository metrics show an anomalous behaviour, the analyzer sends confirmation to the sub-component *Issue Controller*, otherwise the analyzer do not send any notifications and "no alerts" state will be active. The issue controller forwards the measurements to the *Repo-Issue Linker*, when it receives "Yes" signal from the Repo Analyzer, otherwise the controller does not send any data to the linker. The linker uses a Repo-Issue link prediction model to predict links between the repository state and the available issue topics learnt by the Issue Topic Model. The *Reporter* takes the relevant issue topics from the Repo-Issue Linker then sends notifications to the project manager about the possible issues after labeling and ranking them based on the probability of existing links between the issue topics and the repository state which is represented by the measurements taken by the collector. The predicted issues are also sent to the *Issue-Action Linker* which employs the action topic model and Issue-Action link prediction GNN model to predict the action topics. The *Recommender* receives the relevant action topics from the Issue-Action Linker then sends notifications to the project manager about the suggested corrective actions for fixing the reported issues.

3.3.1 Collector and Analyzer

This component is firstly triggered in the framework and used to measure the state of the software repository. It measures the attributes of the artifacts and outputs the values of the repository metrics outlined in Appendix E. The collector also plays an important role in detecting the behaviour of the repository by the sub-component *Repo Analyzer* which detects the abnormal behaviour of the repository and it is based on relating the repository state to one of the clusters acquired after implementing the method explained in detail in Section 3.1.

The analyzer is useful for triggering the other components of the framework. In case there is an anomalous behaviour, the collector will send the new measurement data to the next component which will trigger the issues and also suggest the corrective actions to be taken, otherwise "no alerts" state will be active in the system.

3.3.2 Repo-Issue linker

This component involves running predictions on two ML models, the topic model and the link prediction model. Topic model is trained on extracting the topics among the repository issues and outputs the issue topics given the repository measurements. It is also useful for labeling and generating the description of the predicted issue topics. The linker model which is GNN model trained on predicting the links between the repository measurements and the issue topics. The output of the linker will be the issue topics with probabilities which will be sent to the reporter.

3.3.3 Reporter

The reporter includes only the topic labeling unit in which the title/description of the predicted issue topics are generated and sent to the project manager. We mention that only top issue topics are sent to the manager.

3.3.4 Issue-Action Linker

This component is similar to repo-issue linker and also runs two models, the link prediction model for generating the action topics with respect to the received issue topics from the reporter, and the action topic for getting the action topics learnt during training phase of the models. Topic model is trained on repository commits after classifying the corrective commits using deep neural network based LSTM model.

3.3.5 Recommender

The recommender has the topic labeling unit in which the title/description of the top predicted action topics are generated and sent to the project manager.

3.4 Used Techniques

3.4.1 Clustering Data

To get the initial groupings of repositories, from which we will take the “best” ones, we apply multiple algorithms to avoid being subject to specifics of the math of a single clustering algorithm. We use partitioning-based K-means as one of the generic clustering algorithms (it is applied in many related studies for its simplicity), density-based DBSCAN as more advanced algorithm that allows to identify clusters of arbitrary shapes and handle “noise” in the data and is widely used, and graph-based spectral clustering that can be sought as an intermediate “link” between the two, since it is connected to K-means and to DBSCAN as noted by Dhillon et al. (2004) [113] and Schubert et al. (2018) [114]. The algorithms, their parameters that we use to tune them and their output are presented in Table X. We do not describe the algorithms themselves, as they are very well known.

To internally validate the clustering, we measure three cluster validity

Table X: Clustering algorithms used in the study.

Clustering algorithm	Parameter for tuning	Algorithm's output
K-Means	Number of clusters k	Partition matrix $\boldsymbol{W}$, where each element w_{ik} indicates whether a repository metrics vector $\boldsymbol{x}_i$ belongs to cluster k, and the cluster prototypes (centroids) $\boldsymbol{\mu}_k$
Spectral clustering	Number of clusters k	Partition matrix $\boldsymbol{W}$ and affinity matrix $\boldsymbol{A}$ constructed on software repository metrics vectors $\boldsymbol{x}_i$ [115]
DBSCAN	Neighborhood radius ε	Partition matrix $\boldsymbol{W}$, where noise vectors are given label '-1', and indices j of the core vectors $\boldsymbol{x}_j$ [116]

indices on the clustering results produced by m clustering algorithms run on the metrics data. We use the predefined range of values for such parameters as number of clusters for K-means and Spectral, and ε for DBSCAN. For measuring the indices, we use the Euclidean distance. Since multiple validity indices can show different optimal solutions in terms of the parameter that we tune (e.g., number of clusters for K-means) for the same algorithm, we consider one unique solution per validity index, i.e., if two out of three indices showed the same solution, and the third one showed a new solution, we would consider the two unique solutions; if all three showed the same solution or different solutions, we would consider only one or three unique solutions correspondingly. Due to this reason, on the output of the initial clustering stage, we have $r \geq m$ cluster labelings – one labeling per each unique optimal solution demonstrated by the validity indices.

We apply Silhouette, Calinski-Harabasz, and Davies-Bouldin indices as they are popular in the clustering literature. The following paragraphs briefly describe them.

For each clustered sample $\boldsymbol{x}_i$ of the software repository metrics data, the Silhouette Coefficient s is calculated using the mean intra-cluster distance and the mean nearest-cluster distance (between a sample and the nearest cluster that the sample is not a part of) for each sample. Kaufman and Rousseeuw (2009) [117] introduced the term silhouette coefficient for the maximum value of the mean s over the entire dataset:

$$SC = \max_c \tilde{s}(c), \tag{3.1}$$

where $s(c)$ represents the mean s of the entire dataset for a specific number of clusters c. To find the optimal clustering result, we look for the maximum value of the Silhouette index across the range of considered parameter values and mark a value of a parameter optimal if it corresponds to the maximum value of Silhouette.

Let n_k be the number of datapoints assigned to k-th cluster by a clustering algorithm, $\boldsymbol{\mu}_k$ – prototype (centroid) of k-th cluster, and μ – the global centroid of the data, then Calinski-Harabasz index is calculated using the following formula:

$$CH = \frac{\sum_{k=1}^{c} n_k \cdot \| \boldsymbol{\mu}_k - \boldsymbol{\mu} \|^2}{\sum_{k=1}^{c} \sum_{i=1}^{n_k} \|\boldsymbol{x}_i - \boldsymbol{\mu}_k \;\|^2} \cdot \frac{N-c}{c-1}. \tag{3.2}$$

To identify the optimal clustering result, we look for the maximum value of Calinski-Harabasz index across the range of considered parameter values.

The Davies-Bouldin index is calculated as:

$$DN = \frac{1}{c} \sum_{k=1}^{c} \max_{j \neq k} \frac{(\frac{1}{n_k} \sum_{i=1}^{n_k} \|\boldsymbol{x}_i - \boldsymbol{\mu}_k \|^2)^{\frac{1}{2}} + (\frac{1}{n_j} \sum_{i=1}^{n_j} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j \|^2)^{\frac{1}{2}}}{\| \boldsymbol{\mu}_k - \boldsymbol{\mu}_j \|}. \tag{3.3}$$

To find the optimal clustering result, we look for the minimum value of Davies-Bouldin index across the range of considered parameter values and mark a value of a parameter optimal if it corresponds to the minimum value of Davies-Bouldin.

The values of parameters corresponding to the optimal values of validity indices are saved and then used on the clustering algorithms to cluster the software repository metrics data $\boldsymbol{x}_i$.This clustering then produces the initial optimal cluster labelings λ_q coming from the partition matrices produced by the algorithms, where $q = 1, \ldots, r$, and r is the number of the observed optimal values of parameters across all algorithms (one algorithm can produce up to three optimal cluster labelings, as we use three validity indices).

3.4.2 Consensus Clustering

The application of consensus clustering to the software metrics case had received rather little attention. We were able to identify only few of the studies considering the application of consensus clustering such as those by Coelho et al. (2014) [118] and Puchala et al. (2022) [119]. In addition, we were unable to identify a study that would apply consensus clustering to analyze software repository metrics. In this study, we address that.

Consensus clustering (also called cluster ensembles) provides improved quality of solution and robust clustering as compared to using a single clustering method [120]. The main idea of the consensus clustering approach is to transform the set of individual cluster labelings $\mathbf{\Lambda} = \{\lambda_q \mid q \in 1, \ldots, r\}$ (refer to Section 3.4.1) into a single consensus labeling λ that separates the software repository metrics data into clusters of software repositories using the consensus function:

$$\Gamma : \mathbf{\Lambda} \to \lambda \,. \tag{3.4}$$

In this study, we focus on the graph and hypergraph based as well as non-negative matrix factorization based consensus clustering methods as they are the most popular among the studies related to consensus clustering and are easy to understand and implement [121]. We use the consensus clustering methods (each provides a different consensus function) proposed by Strehl and Ghosh (2002) [108], Fern and Brodley (2004) [122], and Li et al. (2007) [123], namely Cluster-based similarity partitioning algorithm, Hypergraph-partitioning algorithm, Meta-clustering algorithm, Hybrid bipartite graph formulation, and Non-negative matrix factorization, to avoid being subject to the specifics of one consensus function's math, and choose the one that maximizes the average mutual information:

$$\lambda = arg\max_{\hat{\lambda}} \frac{1}{r} \sum_{q=1}^{r} \phi(\hat{\lambda}, \lambda_q), \tag{3.5}$$

where $\phi(\hat{\lambda}, \lambda_q)$ is the normalized mutual information between labelings $\hat{\lambda}, \lambda_q$ and is defined as proposed by Strehl and Ghosh (2002) [108]. The following paragraphs describe the consensus clustering algorithms used.

The idea of Cluster-based similarity partitioning algorithm is to create a square matrix $\boldsymbol{S}$, where each element s_{ij} denotes the fraction of clusterings in which two software repository metrics vectors $\boldsymbol{x}_i, \boldsymbol{x}_j$ are in the same cluster. Next, the similarity matrix is used to recluster the repository metrics vectors using any reasonable similarity-based clustering algorithm, e.g., partitioning of the induced similarity graph (vertex = repository, edge weight = similarity).

In Hypergraph-partitioning algorithm, the consensus clustering problem is formulated as partitioning the hypergraph by cutting a minimal number of hyperedges. All hyperedges are considered to have the same weight, and all vertices are equally weighted. The algorithm looks for a hyperedge separator that partitions the hypergraph into c unconnected components of approximately the same size.

Meta-clustering algorithm is based on clustering clusters of software repositories. Each cluster is represented by a hyperedge. The idea is to group and collapse related hyperedges and assign each repository metrics vector $\boldsymbol{x}_i$ to the collapsed hyperedge in which it participates most strongly. The hyperedges that are considered related for the purpose of collapsing are determined by a graph-based clustering of hyperedges. Each cluster of hyperedges is referred to as a meta-cluster. Collapsing reduces the number of hyperedges from $\sum_{q=1}^{r} c_q$ to c.

Hybrid bipartite graph formulation constructs a graph, where vertices represent software repository clusters or the repository metrics vectors $\boldsymbol{x}_i$. In this graph, cluster vertices are only connected to repository vertices and vice versa, forming a bipartite graph. The algorithm partitions the cluster vertices and the repository vertices of the bipartite graph simultaneously. The partition of the repositories is then outputted as the final clustering.

Nonnegative matrix factorization defines the consensus clustering as the median partition problem, fixing the following distance as a measure of likeness between partitions:

$$d(\lambda, \lambda') = \sum_{i,j=1}^{N} d_{ij}(\lambda, \lambda'),$$

where $d_{ij}(\lambda, \lambda') = 1$ if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to the same cluster in one partition λ and belong to different clusters in the other partition λ', otherwise $d_{ij}(\lambda, \lambda') = 0$. The consensus partition is defined by the median partition problem using d as a dissimilarity measure between partitions.

We use the average normalized mutual information to assess the quality of obtained consensus clustering – we take the consensus that has the largest value of the mutual information and, therefore, the highest quality of obtained clustering.

3.4.3 Identifying Relevant Features

After we aggregated the initial cluster labelings into the consensus cluster labeling λ separating the metrics data into c clusters of software repositories, we estimate a representative vector – "prototype" – for each of the c software repository clusters as the mean across repository metrics vectors $\boldsymbol{x}_i$ in the cluster. We denote a prototype as n-dimensional vector $\boldsymbol{\mu}_k = (\mu_{jk})$, where $k = 1, \dots, c$ – index showing the number of cluster (label) the prototype represents, and $j = 1, \dots, n$ – index showing one of the n metrics.

Our objective is to determine whether the software metrics' values in the prototypes are different across the cluster prototypes $\boldsymbol{\mu}_k$. This leads to the hypothesis:

H_0: μ_{jk} is not statistically significantly different across clusters ($\mu_{j1} = \mu_{j2} = \dots = \mu_{jc}$);

H_1: μ_{jk} is statistically significantly different across clusters ($\exists\, l, t;\ l \neq t : \mu_{jl} \neq \mu_{jt}$).

We conduct ANOVA F-test [124] in case $c > 2$ and t-test when $c = 2$ to check the hypothesis and make conclusion about whether the values of software metric j are statically significantly different across the consensus clusters of software repositories. Then, we repeat this for all n software repository metrics. The set of relevant features consists of the metrics for which we can reject H_0.

3.4.4 Matching Cluster Prototypes and Discrepancy Calculation

At the final step of methodology described in the beginning of Section 3.1, we match the prototypes solving the minimum-weight matching problem in P-partite graph $G = (V, E)$, where vertices $V = v_1, v_2, \dots$ are prototypes produced for P sets of clustered repositories, edges $E = (e_1, e_2, \dots)$ represent and are weighted by the Euclidian distance between prototypes of different repository sets, and each p-th partition represents the set of c prototypes for the p-th repository set; $p = 1, \dots, P$.

We need to find a set of P vertices for each of the c clusters. To solve this problem, we consider all possible combinations of vertices (prototype vectors) from different partitions (P repository sets) and calculate the cost for each combination as the sum of all pairwise distances between prototypes in that combination. The formula for the cost of one combination of i-th prototype of the first repository set, l-th prototype of the second set, ..., h-th prototype of the $(P-1)$-th set, and j-th prototype of the P-th set is the following:

$$C_{\underbrace{il\dots hj}_{P}} = (d_{il} + \dots + d_{ih} + d_{ij}) + (\dots + d_{lh} + d_{lj}) + \dots + (d_{hj}), \tag{3.6}$$

where d_{il} is the Euclidian distance between i-th prototype of the corresponding first set and l-th prototype of the corresponding second set. So, we reformulate the problem as:

$$\min \underbrace{\sum_{i=1}^{c} \sum_{l=1}^{c} \dots \sum_{j=1}^{c}}_{P} C_{il\dots j} X_{il\dots j} \tag{3.7}$$

subject to:

$$\sum_{i=1}^{c}\sum_{l=1}^{c}\dots X_{il\dots j} = 1 \text{ for all } j \in \{1,\dots,c\},$$

$$\dots$$

$$\sum_{i=1}^{c}\dots\sum_{j=1}^{c} X_{il\dots j} = 1 \text{ for all } l \in \{1,\dots,c\},$$

$$\sum_{l=1}^{c}\dots\sum_{j=1}^{c} X_{il\dots j} = 1 \text{ for all } i \in \{1,\dots,c\},$$

$X_{il\dots j} \in \{0,1\}$ for all $\{i,l,\dots,j\} \in \{1,\dots,c\}^P$, where $\{1,\dots,c\}^P$ are disjoint sets of corresponding prototypes of P repository sets. To solve this problem, we use mixed integer linear programming [125], [126].

To make a conclusion about how similar the clusters are across P sets of repositories, we calculate the pairwise discrepancy value d for every set c_i; $i = 1,\dots,c$; consisting of P matched prototypes vectors. For that, we use cosine distance, since it is based on cosine similarity and we want to measure how similar (or dissimilar) the prototype vectors are. However, to not be subject to cosine's distance sensitivity to the mean, we use the adjusted cosine distance:

$$d_{kt} = 1 - \frac{\sum_{i=1}^{n}(\mu_{ik} - \bar{\boldsymbol{\mu}}_k)(\mu_{it} - \bar{\boldsymbol{\mu}}_t)}{\sqrt{\sum_{i=1}^{n}(\mu_{ik} - \bar{\boldsymbol{\mu}}_k)^2}\sqrt{\sum_{i=1}^{n}(\mu_{it} - \bar{\boldsymbol{\mu}}_t)^2}}, \tag{3.8}$$

where μ_{ik} are the elements and $\bar{\boldsymbol{\mu}}_k$ is the mean of the prototype vector $\boldsymbol{\mu}_k$, and d_{kt} measures the adjusted cosine distance between a pair of matched prototypes $\boldsymbol{\mu}_k$ and $\boldsymbol{\mu}_t$ and is bounded in range $[0, 2]$. After measuring the pairwise discrepancy d_{kt} for a set c_i of prototype vectors, we get a single discrepancy value for this set:

$$d_{c_i} = \max_{\boldsymbol{\mu}_k, \boldsymbol{\mu}_t \in c_i} d_{kt}. \tag{3.9}$$

In the end, we will have c discrepancy values d_{c_i}, each corresponding to one of the c clusters. To make sure that we have a meaningful discrepancy values, we repeat the whole algorithm described in Section 3.1 on the random data of the same size $N \times n$ and compare the results with the values achieved in the real data.

3.4.5 Topic modeling on Repository Issues

Since we have diverse issues reported in several repositories, we need a mechanism for grouping the “similar” issues in a single group. We employ the topic modeling techniques which are used extensively on text documents for finding out the topics of the documents in which topic modeling is a helpful step for organizing and categorizing the documents. The objective in topic modeling is to get coherent and diverse topics.

The approach we followed to apply topic modeling on Github issues consists of the following steps:

1. Data Acquisition;
2. Data Preprocessing;
3. Issue Type Classification;
4. Topic Model Training and Evaluation.

The implementation of this approach is described in detail in Section 4.2.1.

Chapter 4

Implementation and Results

This chapter shows the details of the methodology and the building blocks of the proposed framework. We started the implementation of the framework by analyzing and measuring the attributes of the repository in Section 4.1 using Github API. We analyzed the issues of the repositories in Section 4.2.

4.1 Clustering Software Repositories

4.1.1 Data Collection and Preparation

The experimental study was conducted on $N = 970$ GitHub project repositories with $n = 72$ metrics (the full list of them is presented in Appendix E).

The criteria for repository search were the following: number of stars being in range $(100, 150)$, number of forks – in range $(50, 100)$, and size being less than 2100 kilobytes. The number of sets $P = 10$. Following the example of Munaiah et al. (2017) [83] and Pickerill et al. (2020) [89], we took the number of clusters $c = 2$.

4.1.2 Processing of the Repository Sets

First, we normalized each repository set using min-max normalization. In each set of repositories we have identified optimal groupings using three clustering algorithms and three validity indices in Python with mostly Sci-kit learn library by Pedregosa et al. (2011) [127]. The results describing the optimal values of the indices and algorithms' parameters across P sets of repositories are presented in Appendix F.

The consensus clustering algorithms, and their achieved values of the objective function for each of the P repository sets are presented in Table XXII. The highest value of the performance index is underlined for each of the repository sets. The algorithms that correspond to underlined values were used to get a single consensus clustering in each corresponding set of repositories. After getting a single clustering for each of the repository sets, we calculated the prototypes according to our methodology. They were normalized for better visualization and are presented in Figure F.1.

4.1.3 Aggregation of the Results

Since we choose the number of clusters $c = 2$, we conducted the t-test to check the hypothesis H_0 introduced in Section 3.4.3. The t-test was conducted with significance level $\alpha = 0.05$ for metrics based on the clustering results for each of the P sets of clustered repositories separately. Even though for each of the sets the number of "insignificant" metrics varied from 30 to 48, we have identified only 11 metrics for which we have failed to reject the null hypothesis H_0 overall across all P repository sets for all clusters. These metrics were: [Commits] Average per day (True), [Contributors] Count, [Contributors Top-

100] Average additions, [Contributors Top-100] Average deletions, [Contributors Top-100] Average participation week, [Releases] Average assets, [Releases] Average assets downloads, [Releases] Tags, [Repo] Branches, [Repo] Milestones, [Workflow Runs] Average fails per day.

We matched the prototypes from different sets of P clustered repositories using mixed integer programming solver "Coin-or branch and cut" [128] with the Python "puLP" library [129]. We received the following discrepancy values for the two sets of prototypes c_1 and c_2:

$$d_{c_1} \approx 0.164; \; d_{c_2} \approx 0.128.$$

We repeated the whole process with the same values of N, P, n, c for randomly generated data and got the following discrepancy values:

$$d_{c_1} \approx 1.151; \; d_{c_2} \approx 1.172.$$

To check whether our clustering corresponds to the intuition of developers, we prepared a survey by taking the real repository metrics vectors $\boldsymbol{x}_i$ that are the closest to each of the $c \cdot P = 20$ prototypes in terms of Euclidean distance. We call these vectors and corresponding to them repositories the "champions". Since we matched the prototypes and related them to either of $c = 2$ sets each of $P = 10$ prototypes, we have the same separation to 2 sets for champions. Therefore, we can show to the developers these sets of champions: P repositories related to c_1 and the other P related to c_2, and ask their opinion about these sets of repositories to understand whether they really represent two different classes of repositories. The champion repositories corresponding to c_1 are: JeffHoogland/pyxhook, Johankoi/MachoSignature, hagen1778/grafana-import-export, elm/http, thanhniencung/LuckyWheel, naro143/hugo-coder-portfolio, brandicted/scrapy-webdriver, microsoft/winstore-jscompat, OriginalEXE/Switcheroo, Fundoo-Solutions/angularjs-modal-service; and for c_2: chenjr0719/Facebook-Page-Crawler, abusetelegram/telegram-recorder, zzwwjjdj319/miniProgramAmap, jpxue/Overwatch-Aim-Assist, leonvandenbeukel/3D-7-Segment-Digital-Clock, xinghongfei/android-studio-setting, peterWon/CleaningRobot, ultrasonicsoft/ng-animated-login, 0x09AL/DropboxC2C, KyleRicardo/MentoHUST-OpenWrtipk. The results of the survey are discussed in Section 5.2.

Finally, to summarize our clustering across different sets of repositories, we have aggregated the cluster prototypes by taking the mean of P prototypes in each of the c matched prototype sets (the aggregated mean vectors are presented in Figure 4.1 – we normalized them for better visualization) and found the repository metrics vectors $\boldsymbol{x}_i$ that are the closest to these means in terms of Euclidean distance. These repositories' addresses on GitHub are chiuki/android-swipe-image-viewer for the first cluster and jochenwierum/openvpn-manager for the second cluster. The implementation of the whole clustering pipeline is available on https://github.com/kirilldaniakin/RepositoryClusterCaseStudy.

4.2 Repository Issue Analysis

4.2.1 Topic modeling on Repository Issues

Data Acquisition. We initially collected 46986 issues from the 970 repositories used for clustering, then we filtered out the issues that do not indicate

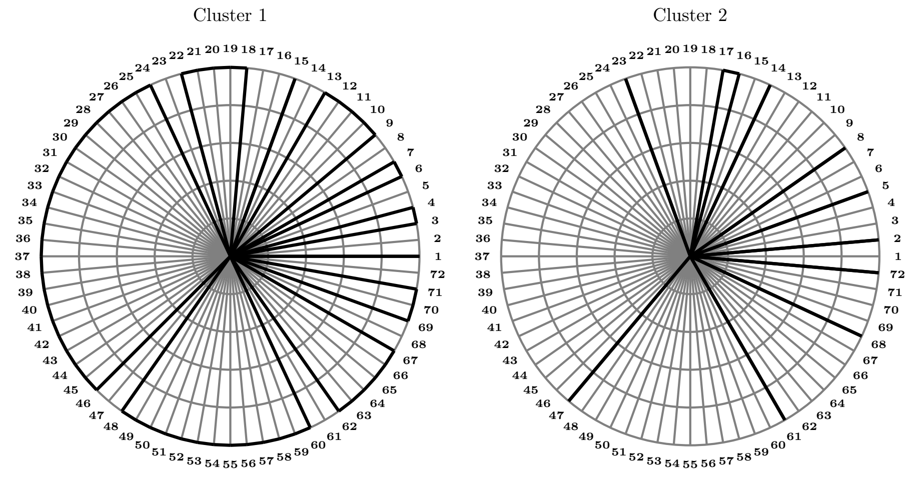


Figure 4.1: Aggregated mean vectors across cluster prototypes.

a problem in the repository. For the sake of this filtering stage, we selected only the issues whose labels included any of the words in the set of "bug" synonyms ("bug", ""error", "fix", "issue", "invalid", "defect", "compatibility", "debt", "refactor"). The selection of the words in this set is based on our exploration of the labels assigned to the issues by developers. The number of preliminary issues included was 2040, but this number was very small compared to the initial number of issues. For this purpose, we built a bug issue detector to detect the issues that indicate bugs, more details are in Section 4.2.2. The Table XI shows the statistics of issue data acquisition step where the column "keyword" represents the keyword extracted from the Github issue, "Repositories" column shows the number of repositories included the respective keyword and "Issues" column represent the number of issues used the respective keyword in one of its labels.

We utilized the NN-based issue classifier whose performance is shown in table XIII, for labeling the issues and it recognized 11417 issues as bug-issues out of 46986 as total, and we run the topic modeling task only on these bug-classified issues.

Table XI: Statistics of issue data acquisition.

Keyword	Repositories	Issues
“bug”	186	1523
“error”	1	4
“fix”	55	140
“issue”	30	88
“invalid”	34	126
“defect”	3	61
“maintenance”	5	31
“debt”	9	40
“refactor”	7	27
Count	2040	211
“*unlabeled*”	914	37329
labeled by our detector	733	11417

Data Preprocessing. The content of the issues includes mostly technical texts, and common text preprocessing heuristics are not sufficient for extracting the candidate words for the topic modeling task. Due to the fact that the issue title and body are submitted by the same reporter to the same issue at the same time, and also the short titles are not sufficient to represent the content of the issue. Due to the previous reasons, we consider the issue title and body as a single unit (issue content) which represents the textual content of the issue. For each issue, we concatenated the issue title and body to get issue content. Since the text here is a technical text, we applied special preprocessing steps as follows:

1. We split the words which are written in camel case, pascal case or snake case;

2. We removed the hyperlinks, tags, inline codes and large codes from the text;

3. We kept all characters in lower-case;
4. We tokenized the text using the NLTK (https://www.nltk.org/) tokenizer;
5. We removed all digits and non-ASCII characters;
6. We also removed the tokens which has a combination of letters and digits;
7. We lemmatized the words using the "WordNetLemmatizer" from NLTK;
8. We removed the single-character tokens;
9. We removed the English stop words provided by NLTK.

The result of this stage is a list of words for each issue text which will be passed as an input to the topic modeling algorithm.

Topic Model Training. There are several techniques for modeling topics of a set of text documents, however little work has been done to perform topic modeling on technical text such as GitHub issues, Stack Overflow discussions, etc. We used the common statistical topic modeling algorithm Latent Dirichlet Allocation (LDA) which has indeed many variations. We used "tomotopy"[1] Python library to conduct topic modeling experiments. As stated in section 2.4, The LDA technique works by learning the distribution of topics in document topic matrix and distribution of words in topic word matrix to maximize the probability that the text corpus is generated by the model. Technically, the model is learnt here using Gibbs sampling. To control the vocabulary generated by the model, we have two hyperparameters which are the minimum collection

[1] https://github.com/bab2min/tomotopy

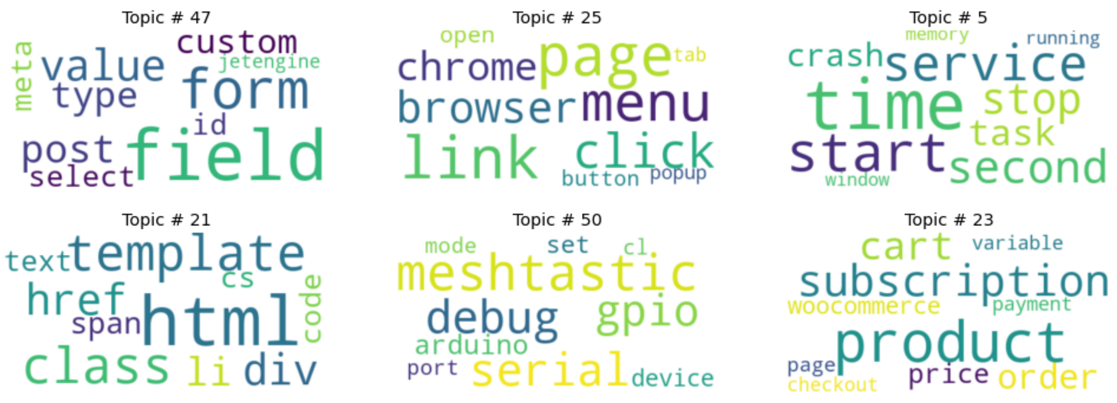


Figure 4.2: Some word clouds sampled from issue topic model.

frequency of words (min_{cf}) and the minimum document frequency of words (min_{df}). Our appropriate settings based on our experiments were $min_{df} = 10$ and $min_{cf} = 10$ where we got the Normalized point-wise Mutual Information (NMPI) [130] value 0.1302. The built vocabulary consisted of 496509 words, 32951 total entries and the entropy of words was 7.18779. By using the previous settings, we built and trained the model using different values for the number of topics K and we got the best score for $K = 64$ topics (Figure 4.3). The extracted topics are presented in table XXV.

Each result topic consists of set of words and we take into consideration the top words which are ranked based on the log-likelihood of the word in the topic. Some of the word clouds are shown in Figure 4.2 where the size of words is proportional to the log-likelihood of the word in the topic.

4.2.2 Issue Type Classification

The motivation for this additional step is due to the reasons stated in section 4.2. In this task, we take the text data of the issues and train a binary classifier to detect the "bug" issues.

For this purpose, we used the dataset published by [112]. The dataset

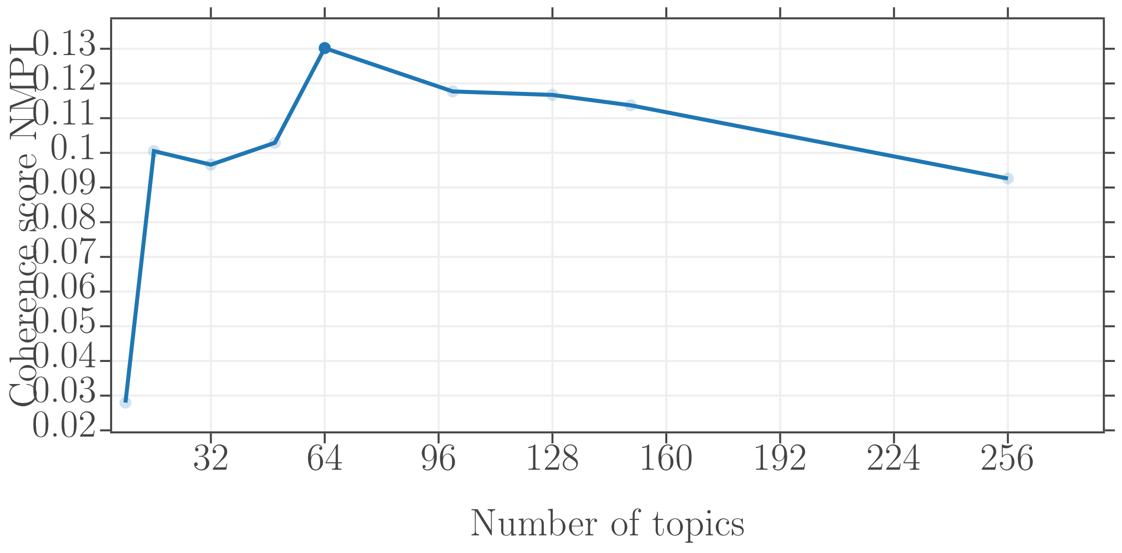


Figure 4.3: Coherence Score NMPI by number of issue topics.

Table XII: Coherence NMPI scores in decreasing order with respect to the number of issue topics.

Number of Topics	NMPI
64	0.1302
100	0.1177
128	0.1167
150	0.1137
50	0.1029
16	0.1005
32	0.0966
256	0.0926
8	0.0280

Table XIII: Issue Classification Results.

Classifier	Text vectorization	Vocabulary size	Accuracy
SVM (kernel = 'RBF')	Skip-gram	200	73.55%
Decision Tree	Skip-gram	200	71.57%
Random Forest	Skip-gram	200	72.33%
NN-based	Trainable NN layer	1000	**81.25%**

is collected using Google BigQuery to query the GitHub Archive. The collected issues were randomly selected from the set of all GitHub issues that were closed in January 2018 and contained one of the labels "bug", "enhancement", or "question". They generated 30,000 issues in which 10,000 issues were labeled as "bug", 10,000 issues were labeled as "enhancement", and 10,000 issues were labeled as "question". Enhancement-labeled issues refer to enhancements and new features. As we mentioned in the beginning of the section, we are interested in this task to detect the issues which indicate bug reports, so we treated the labels "enhancement" and "question" as "non-bug" labels. Eventually, we trained a binary classifier for detecting the "bug" issues. We utilized the same preprocessing steps listed in Section 4.2.1, and Skip-gram technique for vectorizing the textual data of the issues where we used a vocabulary size equals to 200 and the window size was 5. We employed this technique for Support Vector Machine (SVM), decision tree and random forest classifiers. For the neural network (NN) based classifier, we used a trainable text vectorization layer with an embedding layer to learn the latent representations of the input text. The NN-based classifier as shown in Figure 4.4 consists of a bidirectional-LSTM layer followed by two dense layers with "relu" activation and the last output layer has a single unit with "sigmoid" activation. The results of issue classification are presented in Table XIII.

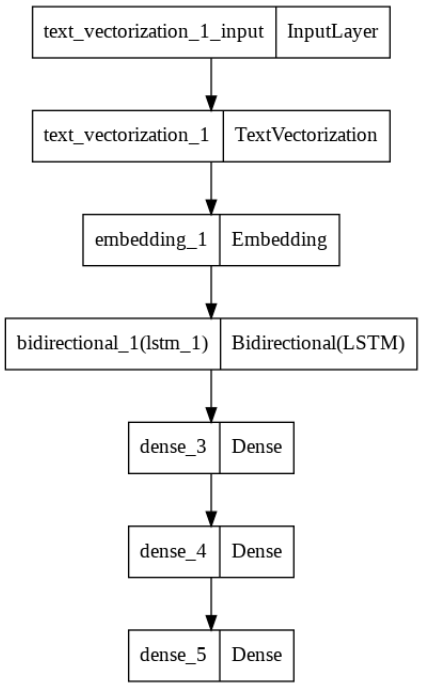

text_vectorization_1_input | InputLayer
text_vectorization_1 | TextVectorization
embedding_1 | Embedding
bidirectional_1(lstm_1) | Bidirectional(LSTM)
dense_3 | Dense
dense_4 | Dense
dense_5 | Dense

Figure 4.4: The architecture of NN-based Bug Issue Detector.

4.2.3 Repo-Issue Link Prediction

The output of the issue topic model is used as an input for the link prediction problem. Here, we built an undirected heterogeneous graph which has two types of nodes ("repo", "issue topic") and a single edge type ("linked"). The graph resembles a bipartite graph in which the repository measurements are linked to issue topics, but each node is a d-dimensional vector. For "repo" node type, we have the 72 metrics as features and for "issue" node type, we have the 100-top words of topics learnt by the issue topic model.

We used HinSAGE implementation of the library StellarGraph [131] to predict the links between these two node types in which the graph is initialized by the ground truth mapping set created as an output of the issue topic model where the repositories are linked to the topics whom their issues belong.

4.3 Repository Commit Analysis

This section describes our analysis work on the data of commits linked to issues classified as bug issues by our classifier in section 4.2.2. The linkage information of the commits to issues are collected by searching for issue events of type “closed”. According to the Github documentation[1], The “closed” event is triggered when the commit message includes a clear statement which refers to the issue number preceded by any of the following words [“fix”, “reslove”, “close”]. At that time, the commit id and commit url will be added as linkage information to the “closed” event of the issues. We managed to extract roughly 500 commits linked to bug issues.

We run the LDA topic modeling on the commit messages and we got the results where we discovered $K = 32$ commit topics with NMPI score being equal to -0.1689. The topics are presented in table XXIII.

[1]https://docs.github.com/en/developers/webhooksandevents/events/issueeventtypes#closed

Chapter 5

Evaluation and Discussion

In this chapter we contextualize our findings from the SLR and the experiments we conducted and discuss the answers to the research questions we outlined in Section 2.1.2.

5.1 Discussion of SLR findings

We collected more than 384 software metrics which were used to predict more than 374 anomalies and to trigger approximately 494 actions. In reviewing our results, we also discovered 112 connections among these three latter data types (metrics, anomalies, actions). These connections comprised 49 metrics, 85 anomalies, and 89 actions. In addition, we also found 233 connections between metrics and anomalies and 250 connections between metrics and actions, as well as 409 anomaly-action connections. Figure 5.1 presents the connections among all the data types we found. In the graph, the anomalies are represented by "red" nodes. "Green" and "blue" nodes display actions and metrics obtained from the data collected. In this graph, an edge exists between metric i and anomaly j nodes if that metric i is used in detecting the anomaly j. The edge between anomaly j and action k nodes means that the action k has been taken to fix or avoid that anomaly j. The edge between metric i and action k nodes indicates the usage of the metric i for triggering the action k or that the action k is triggered by the metric $i's$ measurements.

Our data is available for consultation/download at: [132] . The statistical data is presented in Table XVIII. This information can be used to answer the research questions we outlined in Section 2.1.2.

5.1.1 Analysis of RQ1

In software development, artifact typically refers to "things" that are created, produced, consumed, or modified by the people (developers) involved in the process. Examples include: design documents, workflow diagrams, setup script [133]. Product quality in software development can be evaluated through reference to several quality attributes or metrics that provide quantitative measurements for attributes [134].

It has been shown that software metrics can be useful for estimating, evaluating, and enhancing software quality [135]. Implementing a software metrics program typically aids in the assessment and monitoring and/or identification of improvement actions for achieving quality goals under the ISO/IEC 9126 Standard [136]. Thus, software metrics are useful as much as they provide quantitative methods to measure the quality of a software product or the progress in its development. The adoption of software metrics may also be beneficial for the improvement of the software development process [137]. Regarding this latter point, we wanted to gather the metrics that are currently used to predict defects or anomalies, as well as those that are used to recommend actions for fixing those anomalies and defects.

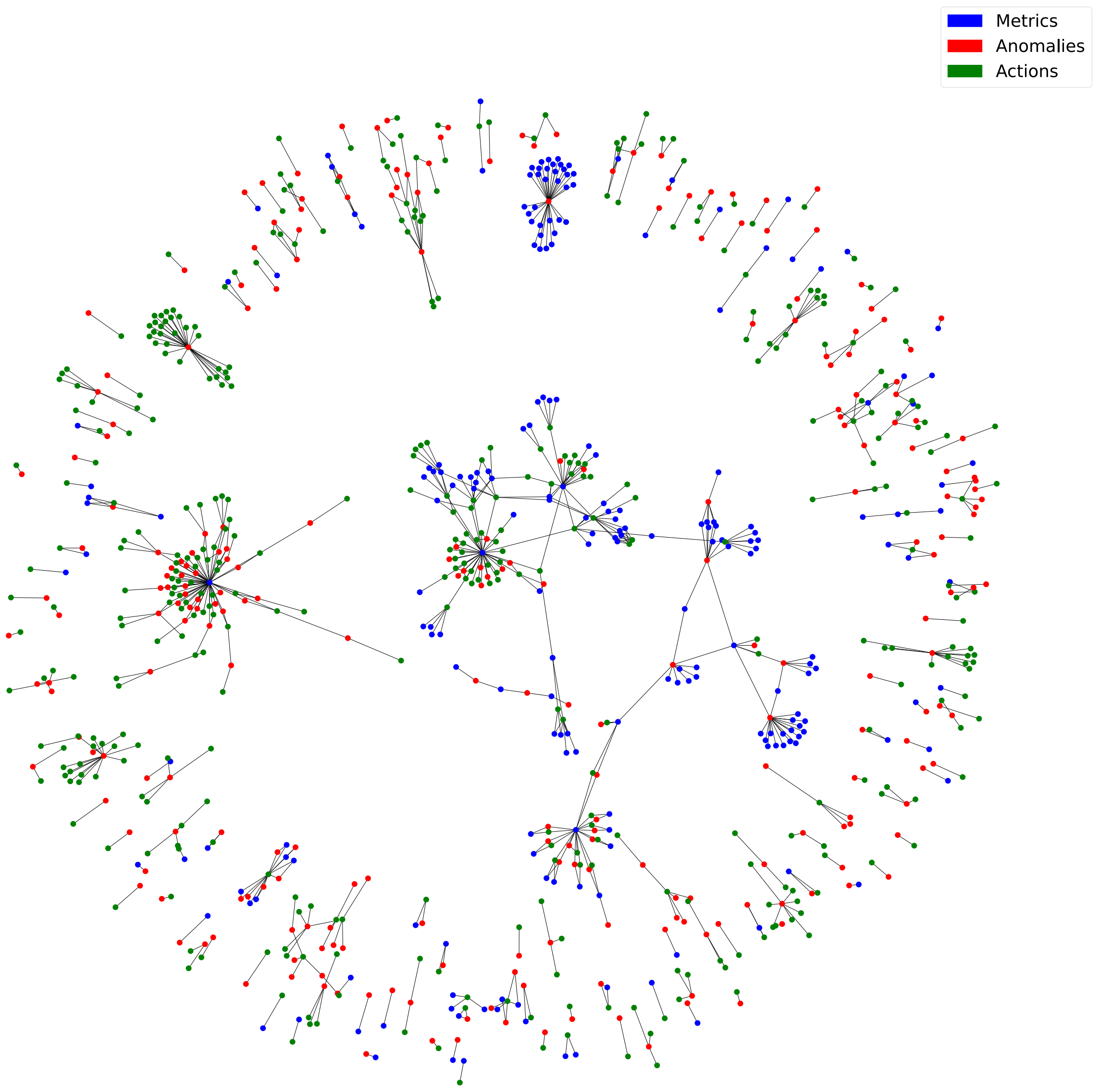


Figure 5.1: Undirected graph representing the connections between the data types.

As shown in Chapter 4, we collected a large number of metrics that are typically used for multiple purposes. Specifically, we gathered -across the 42 studies that provided software metrics data- 231 metrics that were used for detecting 124 software anomalies or for triggering 141 specific actions. Even though we presented in Table IV the metrics that are mostly used in the studies we selected, we were unable to find a common metric among all the included studies. Based on the data we gathered, we can say that "Lines Of Code" was used in 12 out of 42 studies. Likewise, "Cyclomatic Complexity", was reported as being used in 7 studies. This shows that researchers are not consistent with metrics selection and that there is a lack of a common glossary in the field as also noted by Daniakin (2021) [138]. This seems to be a significant result for the field; however, at present we don't have a well-formed scientific explanation for it.

5.1.2 Analysis of RQ2

This research question attempted to specify and classify the metrics most frequently used for detecting or predicting software anomalies. In this SLR, we were interested in spotting and classifying anomalies across different software development processes. In this work, we did not focus on detecting specific anomalies just in the source code (code smells). In Section 2.1.4, we preliminarily presented the results we gathered on 42 studies (which used software metrics) and on 45 studies (which reported anomalies). We then observed that only 27 studies among those we included in the final log discussed the usage of metrics for detecting anomalies. In these studies, we found that over 155 metrics were used for detecting 124 anomalies.

We also discovered that the anomaly detected by the largest number of metrics is “CI build failure” [77] as shown in Table VIII. In addition, we note that “defect injection metric” was the most useful indicator for detecting 30 anomalies and the metrics “defect density” and “frequency of interactions / number of meetings” were used for predicting 10 anomalies each. This data is presented in Table V. We also acknowledge that the anomalies presented in this SLR are uniquely reported in the studies we reviewed. Likewise, we should emphasise that we couldn’t reach a common data point for the anomaly data found in our studies. This indicates that there is an enormous gap in the field from the perspective of specifying the anomalies encountered in software development projects, and also a significant gap in providing a common action plan for handling them.

5.1.3 Analysis of RQ3

Nevertheless, metrics are not sufficient for predicting anomalies, at least until they are fixed [69]. Anomalies get fixed by applying a set of suggested corrective actions followed by the execution of preventive actions. This is done to avoid their recurrence in the future. Recommended actions are used in specific scenarios, and, to implement them, one typically needs to trace back anomalies until their root causes are individuated. As shown in Section 2.1.4, we collected a significant amount of 360 actions suggested for 224 anomalies. In particular, we observed in Table VII that the software anomalies that had the greatest number of action connections were “class resistance to change” (29 actions developed) and “inaccurate estimation of efforts” (15 actions recommended).

5.1.4 Analysis of RQ4

As we showed in Section 2.1.4, metrics can be classified into metrics used for detecting anomalies and metrics used for triggering actions. The metrics used for triggering actions are called corrective/preventive metrics and are used for fixing/avoiding software anomalies during software maintenance [139]. In this work, we did not cover only the anomalies or defects that may arise after deploying a software, but tried to cover all sorts of software anomalies as they may arise in any process of the software development life cycle. Specifically, we found 125 metrics used for providing 141 action recommendations. We also observed that "defect injection" is the metric used for triggering most actions (30); whereas "defect density" is responsible for triggering 23 actionsas shown in Table VI. We can also point out for the reader that the action "train developers to understand the requirement errors" was triggered by the largest number of metrics, as shown in Table IX. It is important to note that the actions are unique and that the Table IX does not include enhancement actions, but only preventive and corrective actions. The column "Used Metrics Count" in Table IX represents the number of metrics used for triggering the corresponding action. The column "Reported in Studies" contains the study that reported the action presented in the column "Anomaly". In this context, we should note that we could not find a common action across all the studies we reviewed. Since the studies uniquely reported anomalies and did not properly classify them, the actions were contextual and specific to individual anomalies.

5.1.5 A Synoptic Summary

This SLR aimed to study and find out how to recommend actions for correcting or preventing anomalies detected in software projects automatically. In other words, our research attempted to build connections between quantitative measurements of software attributes or metrics and anomalies in order to suggest beneficial managerial actions. The outcomes of this research can be summarised as follows:

- There is a large number of metrics in the literature that are being used for software anomaly detection;
- Most of the metrics we found are used for detecting or preventing software defects/anomalies;
- Fewer metrics are used for triggering actions;
- The impact of those actions is often not properly measured or is studied in a relatively small number of studies;
- We did not find any study that automated the process of recommending actions by using metrics;
- Root causes of anomalies play important roles in individuating appropriate corrective and preventive actions;
- The majority of actions are suggested for managing software projects or finding technical solutions in accordance with CMMI key process areas [81].
- We did not find a common anomaly or action across the studies.

We focused on corrective actions in our framework, since preventive actions require root cause analysis aimed at getting rid of the root of an anomaly, which is harder to do automatically.

5.2 Discussion on Clustering Software Repositories

Since the adjusted cosine distance, used to calculate the discrepancy values, is bounded in the range $[0, 2]$, we make a conclusion that the discrepancy values d_{c_1} and d_{c_2} (refer to Section 4.1.3) achieved on the real data signal of consistent clustering across the subsets of data as opposed to the discrepancy on the random data (refer to Section 4.1.3). This means that there indeed might be an underlying structure in the form of two clusters across our repositories.

We have conducted a survey with 20 computer science students of Innopolis University, who have the experience in software development, to check whether our clustering corresponds to the intuition of the developers. We showed them the 20 champion repositories mentioned in Section 4.1.3 and for each repository asked whether they would contribute to the repository or not, and if they answered no, we asked why. According to our results, the repositories from the first cluster c_1 got the majority of positive answers, while the second cluster c_2 got less agreeable respondents' answers, where three out of ten repositories received around 50% of both positive and negative answers. We believe this demonstrates that our clustering was relatively successful from the perspective of developers' intuition, since we marked the first cluster as "maintained" repositories and the second – as "less maintained", and people tend to be more puzzled in understanding less maintained repositories. We marked the second cluster as "less maintained" rather than "not maintained", since for our analysis we used relatively successful projects according to the popularity/size ratio (refer to Section 4.1.1).

We have conducted the study without filtering out the GitHub repositories with non-English descriptions and Readme's, and, after conducting the survey with developers, it turned out that all of the champions with Chinese Readme's got negative results (the developers answered no and, as the reason, mentioned that they do not understand what is going on in the repository because of the Chinese language). We believe that this introduced a certain bias, because for the rest of the survey the developers were biased to answer yes to the repositories with English descriptions and Readme's. In addition, we did not filter the repositories by programming language and, again, got biased results, where the developers said that they would not contribute simply because they do not use the repository's programming language.

Concerning the relevant features for clustering, we have identified only 11 metrics (out of 72) that did not differ statistically significantly across the repository clusters. Even though we on purpose narrowed down the range of the repository size and the number of stars and forks, according to the t-test, these metrics still vary across the clusters, which potentially signals about how dominant these metrics are compared to the others that we used. In addition, after our analysis, we were still left with 61 metric, which is quite a significant amount for practitioners to comprehend for manual repository analysis.

To find software repositories for our analysis, we used a rather strict search criteria that narrowed down the list of repositories to approximately the same size and popularity, however, if we did not use such criteria, our clustering would be with high probability affected by the metrics related to forks, stars, number of contributors, etc., i.e., by the metrics relating to size and popularity, which would not present any use or interest to practitioners as such size and popularity-wise classification can be done easily by hand.

In terms of external validity (i.e., the extent to which the results of a study can be generalized to other situations), we cannot generalize our findings to the development platforms other than GitHub. However, GitHub now hosts more than 19.4 million active repositories, making it a good starting point for this research [49].

In terms of construct validity (i.e., the degree to which a test measures what it claims, or purports, to be measuring), while our initial list of software repository metrics was designed to be comprehensive and based on most used and provided by GitHub API, there could be other important metrics that we did not include in the initial set of repository metrics. However, our proposed approach for discovering the underlying groups of repositories can be used with different set of metrics including code metrics, which could provide a deeper insight into repositories' code structure.

The resulting prototypes for the two clusters presented in Figure 4.1 can be used to classify a new repository that have not been clustered before by relating it to one of the two clusters. In case the new repository is closer (according to, e.g., Euclidean distance) to the ”less maintained” cluster, we identify the repository's metrics whose values differ significantly from the mean of those encountered in the ”maintained” cluster (in other words, we compare the new repository to the prototype of the maintained cluster) and based on these metrics we recommend corrective actions that would help to move the repository to the maintained cluster.

The approach of aggregating the clustering results for different sets of repositories presented here is similar to the meta-analysis techniques and, in fact, can be used in conjunction with the method proposed by Daniakin (2021) [138]. For example, if we had multiple clusters of studies with calculated meta-analytical effect size for each of the clusters received after applying the methodology from [138], we could compare the results across the clusters by calculating the discrepancy values, and then compare them to the values calculated on a randomly generated data for validation, as proposed in this work.

5.3 Testing the Proposed Framework

As the diagram shows in figure 3.1, the framework consisted of multiple components and in order to test the framework as a whole, we selected 10 random Github repositories which have the same characteristics outlined in section 4.1.1. The "repo analyzer" labeled 6 repositories as normal and *no alerts* signal is raised, whereas four of them are categorized as *unusual* repositories. The evaluation results are presented in table XV where the "unusual" repositories further went through the other components of the framework to predict the possible issue topics and recommend corrective actions.

The issue topics predicted by the "Repo-Issue Linker", labeled by the "Reporter" and the action topics predicted by the "Issue-Action Linker", labeled by the "Recommender" are presented in table XV. The issue and action/commit topics are shown in table XXV and table XXIII, respectively.

We conclude from the results shown in table XXVI and table XXIV that the labels cannot be generalized to different applications and multiple phases of software development. This is due to the fact that the issue contents include more technical and less descriptive texts and also the commit messages are short

Repository	Is it unusual
akDeveloper/Aktive-Merchant	no
dogriffiths/HeadFirstC	no
openwisp/openwisp-users	yes
intro-to-ml-with-kubeflow/intro-to-ml-with-kubeflow-examples	no
fsprojects/Paket.VisualStudio	yes
Iceloof/GoogleNews	no
PacktPublishing/Hands-On-Software-Architecture-with-Golang	no
CalebCurry/python	no
mampfes/hacs_waste_collection_schedule	yes
uos/rospy_message_converter	yes

Table XIV: Evaluation results of the proposed framework

Repository	Predicted issue topics	Suggested Action topics
openwisp/openwisp-users	44	14
	40	31
	47	5
fsprojects/Paket.VisualStudio	61	30
	50	[31, 19]
	44	17
	43	17
mampfes/hacs_waste_collection_schedule	3	22
	62	19
uos/rospy_message_converter	19	23
	44	16

Table XV: Issue topics predicted and corrective action topics suggested for the issues

and explain actions taken to fix or correct an issue but they are less descriptive too.

Chapter 6

Conclusion

In this work we have investigated the industrial challenge related to managing software development issues and proposed the methodology to attribute software metrics to corrective actions. We have conducted a systematic literature review that highlighted the research gaps in the field of software repository analysis, results of which were used to build a framework of detecting software repository anomalies and recommending corrective actions to address the anomalies. We have implemented and validated the framework on 10 randomly selected GitHub repositories. We have got 64 anomalies corresponding to the 32 actions, and for the "anomalous" repositories, our framework have provided the corrective actions.

This study is intended to be the first of many works to explore the application of machine learning approaches for recommending corrective actions to software development projects. This work presented the use of modern technologies to explore the issues in managing software development. Hence, this work represents an attempt towards creating more efficient anomaly detection and action recommendation tools grounded in research.

For the future work, we are considering building metric-based corrective and preventive action recommendation system which supports GitLab and Jira, applying different interpretable dimensionality reduction techniques to reduce the number of software repository metrics to a human-comprehendable number, using code metrics to get deeper insights of what is going on inside the repositories, repeating the data collection, clustering, and the survey process including only English and a single programming language (e.g. Java) GitHub repositories, and conducting industrial experiments to collect more relevant data than what is published in GitHub repositories. In addition, we are considering the use of semi-supervised clustering (e.g., COP K-Means [140]) to incorporate the knowledge about the repositories that we acquired from the survey into our clustering to potentially improve the accuracy of division of the software repositories.

Bibliography cited

[1] Standish Group, "The Chaos Report," *United States of America*, 2015.

[2] M. H. N. Nasir and S. Sahibuddin, "Critical Success Factors for Software Projects: A Comparative Study," *Scientific Research and Essays*, vol. 6, no. 10, pp. 2174–2186, 2011, ISSN: 1992-2248.

[3] R. N. Charette, "Why Software Fails [Software Failure]," *IEEE Spectrum*, vol. 42, no. 9, pp. 42–49, 2005, ISSN: 0018-9235. DOI: 10.1109/MSPEC.2005.1502528.

[4] N. Cerpa and J. M. Verner, "Why Did Your Project Fail?" *Communications of the ACM*, vol. 52, no. 12, pp. 130–134, 2009, ISSN: 0001-0782. DOI: 10.1145/1610252.1610286. [Online]. Available: https://doi.org/10.1145/1610252.1610286.

[5] M. Cataldo and J. D. Herbsleb, "Coordination Breakdowns and Their Impact on Development Productivity and Software Failures," *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp. 343–360, 2012, ISSN: 0098-5589. DOI: 10.1109/TSE.2012.32.

[6] K. A. Demir, "A Survey on Challenges of Software Project Management.," in *Proceedings of the 2009 International Conference on Software Engineering Research & Practice, SERP 2009, July 13-16, 2009,*

Las Vegas, Nevada, USA, 2 Volumes, H. R. Arabnia and H. Reza, Eds., vol. 2009, CSREA Press, 2009, pp. 579–585.

[7] M. Kuutila, M. Mäntylä, U. Farooq, and M. Claes, "Time Pressure in Software Engineering: A Systematic Review," *Information and Software Technology*, vol. 121, p. 106 257, 2020, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2020.106257. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920300045.

[8] J. Verner, J. Sampson, and N. Cerpa, "What Factors Lead to Software Project Failure?" In *Proceedings of the 2008 second international conference on research challenges in information science*, IEEE, 2008, pp. 71–80. DOI: 10.1109/RCIS.2008.4632095.

[9] L. Kappelman, R. McKeeman, and L. Zhang, "Early Warning Signs of It Project Failure: The Dominant Dozen," *Information Systems Management*, vol. 23, no. 4, pp. 31–36, 2006, ISSN: 0736-6981. DOI: 10.1201/1078.10580530/46352.23.4.20060901/95110.4. eprint: https://doi.org/10.1201/1078.10580530/46352.23.4.20060901/95110.4. [Online]. Available: https://doi.org/10.1201/1078.10580530/46352.23.4.20060901/95110.4.

[10] T. O. Lehtinen, M. V. Mäntylä, J. Vanhanen, J. Itkonen, and C. Lassenius, "Perceived Causes of Software Project Failures–an Analysis of Their Relationships," *Information and Software Technology*, vol. 56, no. 6, pp. 623–643, 2014, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2014.01.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584914000263.

[11] D. Tesch, T. J. Kloppenborg, and M. N. Frolick, "IT Project Risk Factors: The Project Management Professionals Perspective," *Journal of*

Computer Information Systems, vol. 47, no. 4, pp. 61–69, 2007, ISSN: 8756-9728.

[12] S. McConnell, "Avoiding Classic Mistakes [Software Engineering]," *IEEE Software*, vol. 13, no. 5, p. 112, 1996, ISSN: 0740-7459. DOI: 10.1109/52.536469.

[13] B. Whittaker, "What Went Wrong? Unsuccessful Information Technology Projects," *Information Management & Computer Security*, vol. 7, pp. 23–30, 1999, ISSN: 0968-5227. [Online]. Available: https://doi.org/10.1108/09685229910255160.

[14] P. L. Bannerman, "Risk and Risk Management in Software Projects: A Reassessment," *Journal of Systems and Software*, vol. 81, no. 12, pp. 2118–2133, 2008, Best papers from the 2007 Australian Software Engineering Conference (ASWEC 2007), Melbourne, Australia, April 10-13, 2007, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2008.03.059. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121208000897.

[15] H. Taherdoost, "How to Lead to Sustainable and Successful IT Project Management? Propose 5Ps Guideline," *Propose 5Ps Guideline (August 1, 2018)*, 2018. eprint: https://dx.doi.org/10.2139/ssrn.3224225.

[16] D. Moitra, "Managing Organizational Change for Software Process Improvement," in *Software Process Modeling*, S. T. Acuña and N. Juristo, Eds. Boston, MA: Springer US, 2005, pp. 163–185. DOI: 10.1007/0-387-24262-7_7. [Online]. Available: https://doi.org/10.1007/0-387-24262-7_7.

[17] N. E. Fenton and M. Neil, "Software Metrics: Successes, Failures and New Directions," *Journal of Systems and Software*, vol. 47, no. 2-3, pp. 149–157, 1999, ISSN: 0164-1212. DOI: 10.1016/S0164-1212(99)00035-7. [Online]. Available: https://doi.org/10.1016/S0164-1212(99)00035-7.

[18] K. E. Emam, W. Melo, and J. C. Machado, "The Prediction of Faulty Classes Using Object-oriented Design Metrics," *Journal of Systems and Software*, vol. 56, no. 1, pp. 63–75, 2001, ISSN: 0164-1212. DOI: 10.1016/S0164-1212(00)00086-8. [Online]. Available: https://doi.org/10.1016/S0164-1212(00)00086-8.

[19] G. P. Bhandari and R. Gupta, "Measuring the Fault Predictability of Software Using Deep Learning Techniques with Software Metrics," in *Proceedings of the 2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, IEEE, 2018, pp. 1–6. DOI: 10.1109/UPCON.2018.8597154.

[20] Y. Hu, X. Mo, X. Zhang, Y. Zeng, J. Du, and K. Xie, "Intelligent Analysis Model for Outsourced Software Project Risk Using Constraint-based Bayesian Network," *Journal of Software*, vol. 7, no. 2, pp. 440–449, 2012, ISSN: 1796-217X.

[21] K. Mordal, N. Anquetil, J. Laval, A. Serebrenik, B. Vasilescu, and S. Ducasse, "Software Quality Metrics Aggregation in Industry," *Journal of Software: Evolution and Process*, vol. 25, no. 10, pp. 1117–1135, 2013. DOI: https://doi.org/10.1002/smr.1558. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1558. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1558.

[22] D. M. Martínez and J. C. Fernandez-Rodriguez, “Artificial Intelligence Applied to Project Success: A Literature Review,” *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 3, no. 5, pp. 77–84, 2015, ISSN: 1989-1660.

[23] W. M. Zage, D. M. Zage, and C. Wilburn, “Avoiding Metric Monsters: A Design Metrics Approach,” *Annals of Software Engineering*, vol. 1, no. 1, pp. 43–55, 1995, ISSN: 1022-7091. DOI: 10.1007/BF02249045. [Online]. Available: https://doi.org/10.1007/BF02249045.

[24] R. Noor and M. F. Khan, “Defect Management in Agile Software Development,” *International Journal of Modern Education and Computer Science*, vol. 6, no. 3, p. 55, 2014, ISSN: 2075-0161. DOI: 10.5815/ijmecs.2014.03.07.

[25] V. Suma and T. G. Nair, “Effective Defect Prevention Approach in Software Process for Achieving Better Quality Levels,” *arXiv preprint arXiv:1001.3552*, vol. 6, 2010, ISSN: 1746-6474.

[26] N. S. Gill, “Factors Affecting Effective Software Quality Management Revisited,” *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 2, pp. 1–4, 2005, ISSN: 0163-5948. DOI: 10.1145/1050849.1050862. [Online]. Available: https://doi.org/10.1145/1050849.1050862.

[27] A. F. Dareshuri, E. F. Darehshori, A. H. Hardoroudi, and H. M. Sarkan, “Implementing Corrective and Preventive Actions in Risk Assessment Software,” in *Proceedings of the 2011 IEEE Conference on Open Systems*, IEEE, 2011, pp. 327–331. DOI: 10.1109/ICOS.2011.6079271.

[28] A. H. Hardoroudi, A. F. Dareshuri, H. M. Sarkan, and M. Nourizadeh, “Robust Corrective and Preventive Action (CAPA),” in *Proceedings of*

the 2011 IEEE International Systems Conference, IEEE, 2011, pp. 177–181. DOI: 10.1109/SYSCON.2011.5929081.

[29] S. Marcos-Pablos and F. J. Garcıéa-Peñalvo, "Decision support tools for slr search string construction," in *Proceedings of the sixth international conference on technological ecosystems for enhancing multiculturality*, 2018, pp. 660–667.

[30] B. Kitchenham, "Procedure for Undertaking Systematic Reviews," Computer Science Department, Keele University (TRISE-0401) and National ICT Australia Ltd (0400011T. 1), Joint Technical Report, Tech. Rep., 2004.

[31] B. Kitchenham and S. Charters, "Guidelines for Performing Systematic Literature Reviews in Software Engineering," Keele University and Durham University Joint Report, Tech. Rep., 2007. [Online]. Available: https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf.

[32] B. Kitchenham, P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic Literature Reviews in Software Engineering–a Systematic Literature Review," *Information and Software Technology*, vol. 51, no. 1, pp. 7–15, 2009, Special Section - Most Cited Articles in 2002 and Regular Research Papers, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2008.09.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584908001390.

[33] P. Brereton, B. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain," *Journal of Systems and Soft-*

ware, vol. 80, no. 4, pp. 571–583, 2007, Software Performance, ISSN: 0164-1212. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016412120600197X.

[34] M. Farina, A. Gorb, A. Kruglov, and G. Succi, “Technologies for gqm-based metrics recommender systems: A systematic literature review,” *IEEE Access*, vol. 10, pp. 23 098–23 111, 2022. DOI: 10.1109/ACCESS.2022.3152397.

[35] C. Differding, B. Hoisl, and C. M. Lott, “Technology Package for the Goal Question Metric Paradigm,” Fachbereich Informatik, Tech. Rep. 281, 2017, p. 27. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-49169.

[36] M. J. Page, J. E. McKenzie, P. M. Bossuyt, *et al.*, “Updating Guidance for Reporting Systematic Reviews: Development of the PRISMA 2020 Statement,” *BMJ*, vol. 134, pp. 103–112, 2021, ISSN: 0895-4356. DOI: 10.1136/bmj.n71. eprint: https://www.bmj.com/content/372/bmj.n71.full.pdf.

[37] R. H. Thayer, A. Pyster, and R. C. Wood, “Major Issues in Software Engineering Project Management,” *IEEE Transactions on Software Engineering*, vol. SE-7, no. 4, pp. 333–342, 1981, ISSN: 0098-5589. DOI: 10.1080/08874417.2007.11645981. eprint: https://www.tandfonline.com/doi/pdf/10.1080/08874417.2007.11645981. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/08874417.2007.11645981.

[38] M. Caulo and G. Scanniello, “A Taxonomy of Metrics for Software Fault Prediction,” in *Proceedings of the 2020 46th Euromicro Conference on*

Software Engineering and Advanced Applications (SEAA), IEEE, 2020, pp. 429–436. DOI: 10.1109/SEAA51224.2020.00075.

[39] L. Kumar, S. Misra, and S. K. Rath, "An Empirical Analysis of the Effectiveness of Software Metrics and Fault Prediction Model for Identifying Faulty Classes," *Computer Standards & Interfaces*, vol. 53, pp. 1–32, 2017, ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2017.02.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920548916300885.

[40] G. Abaei, A. Selamat, and J. A. Dallal, "A Fuzzy Logic Expert System to Predict Module Fault Proneness Using Unlabeled Data," *Journal of King Saud University - Computer and Information Sciences*, vol. 32, no. 6, pp. 684–699, 2020, ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2018.08.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157818300247.

[41] S. M. Nassar, "Software Reliability," *Computers & Industrial Engineering*, vol. 11, no. 1-4, pp. 613–618, 1986, ISSN: 0360-8352. DOI: 10.1016/0360-8352(86)90164-6. [Online]. Available: https://doi.org/10.1016/0360-8352(86)90164-6.

[42] B. Kitchenham and J. Walker, "A Quantitative Approach to Monitoring Software Development," *Software Engineering Journal*, vol. 4, no. 1, pp. 2–13, 1989, ISSN: 0268-6961. DOI: 10.1049/sej.1989.0001. [Online]. Available: https://doi.org/10.1049/sej.1989.0001.

[43] D. N. Card, "Software Product Assurance: Measurement and Control," *Information and Software Technology*, vol. 30, no. 6, pp. 322–330, 1988, The Software Life Cycle Part II, ISSN: 0950-5849. DOI: https://doi.

org/10.1016/0950-5849(88)90010-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0950584988900109.

[44] A. A. Porter and R. W. Selby, "Empirically Guided Software Development Using Metric-based Classification Trees," *IEEE Software*, vol. 7, no. 2, pp. 46–54, 1990, ISSN: 0740-7459. DOI: 10.1109/52.50773.

[45] J. A. Lane and D. Zubrow, "Integrating Measurement with Improvement: An Action-Oriented Approach: Experience Report, An action-oriented approach: Experience report," in *Proceedings of the 19th International Conference on Software Engineering*, 1997, pp. 380–389. DOI: 10.1109/ICSE.1997.610293.

[46] C. Ebert, "Technical Controlling and Software Process Improvement," *Journal of Systems and Software*, vol. 46, no. 1, pp. 25–39, 1999, ISSN: 0164-1212. DOI: 10.1016/S0164-1212(98)10086-9. [Online]. Available: https://doi.org/10.1016/S0164-1212(98)10086-9.

[47] L. Kapová, T. Goldschmidt, S. Becker, and J. Henss, "Evaluating Maintainability with Code Metrics for Model-to-model Transformations," in *Proceedings of the International Conference on the Quality of Software Architectures*, G. T. Heineman, J. Kofron, and F. Plasil, Eds., Springer Berlin Heidelberg, 2010, pp. 151–166, ISBN: 978-3-642-13821-8.

[48] I. Chowdhury and M. Zulkernine, "Using Complexity, Coupling, and Cohesion Metrics As Early Indicators of Vulnerabilities," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011, Special Issue on Security and Dependability Assurance of Software Architectures, ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2010.06.003. [On-

line]. Available: https://www.sciencedirect.com/science/article/pii/S1383762110000615.

[49] C. Treude, L. Leite, and M. Aniche, "Unusual events in github repositories," *Journal of Systems and Software*, vol. 142, pp. 237–247, 2018, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2018.04.063. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121218300876.

[50] L. Zong, "Classification Based Software Defect Prediction Model for Finance Software System-an Industry Study," in *Proceedings of the 2019 3rd International Conference on Software and e-business*, 2019, pp. 60–65.

[51] N. E. Fenton and M. Neil, "Software Metrics: Roadmap, Roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, ser. ICSE '00, Limerick, Ireland: Association for Computing Machinery, 2000, pp. 357–370. DOI: 10.1145/336512.336588. [Online]. Available: https://doi.org/10.1145/336512.336588.

[52] Y. Shi, M. Li, S. Arndt, and C. Smidts, "Metric-based Software Reliability Prediction Approach and Its Application," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1579–1633, 2017, ISSN: 1382-3256. DOI: 10.1007/s10664-016-9425-9. [Online]. Available: https://doi.org/10.1007/s10664-016-9425-9.

[53] M. Korkala and F. Maurer, "Waste Identification As the Means for Improving Communication in Globally Distributed Agile Software Development," *Journal of Systems and Software*, vol. 95, pp. 122–140, 2014, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2014.03.080. [On-

line]. Available: https://www.sciencedirect.com/science/article/pii/S0164121214001009.

[54] M. Lavallee and P. N. Robillard, "Why Good Developers Write Bad Code: An Observational Case Study of the Impacts of Organizational Factors on Software Quality," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15, vol. 1, Florence, Italy: IEEE Press, 2015, pp. 677–687, ISBN: 9781479919345.

[55] T. D. Oyetoyan, D. Cruzes, and R. Conradi, "A Study of Cyclic Dependencies on Defect Profile of Software Components," *Journal of Systems and Software*, vol. 86, no. 12, pp. 3162–3182, 2013, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2013.07.039. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121213001878.

[56] S. Kolahdouz-Rahimi, K. Lano, and M. Karimi, "Technical Debt in Procedural Model Transformation Languages," *Journal of Computer Languages*, vol. 59, p. 100 971, 2020, ISSN: 2590-1184. DOI: https://doi.org/10.1016/j.cola.2020.100971. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590118420300319.

[57] M. Tsunoda, T. Matsumura, and K.-i. Matsumoto, "Modeling Software Project Monitoring with Stakeholders," in *Proceedings of the 2010 IEEE/ACIS 9th International Conference on Computer and Information Science*, IEEE, 2010, pp. 723–728. DOI: 10.1109/ICIS.2010.84.

[58] S. Kumaresh and R. Baskaran, "Defect Analysis and Prevention for Software Process Quality Improvement," *International Journal of Computer Applications*, vol. 8, no. 7, pp. 42–47, 2010, ISSN: 0975-8887. DOI: 10.5120/1218-1759.

[59] D. F. Brito, M. P. Barcellos, and G. Santos, "Investigating Measures for Applying Statistical Process Control in Software Organizations," *Journal of Software Engineering Research and Development*, vol. 6, no. 1, pp. 1–31, 2018, ISSN: 2195-1721. DOI: 10.1186/s40411-018-0054-4.

[60] A. A. Shenvi, "Defect Prevention with Orthogonal Defect Classification," in *Proceedings of the 2nd India Software Engineering Conference*, ser. ISEC '09, Pune, India: Association for Computing Machinery, 2009, pp. 83–88, ISBN: 9781605584263. DOI: 10.1145/1506216.1506232. [Online]. Available: https://doi.org/10.1145/1506216.1506232.

[61] I. Bhandari, M. Halliday, E. Traver, D. Brown, J. Chaar, and R. Chillarege, "A Case Study of Software Process Improvement during Development," *IEEE Transactions on Software Engineering*, vol. 19, no. 12, pp. 1157–1170, 1993. DOI: 10.1109/32.249661.

[62] M. Leszak, D. E. Perry, and D. Stoll, "Classification and Evaluation of Defects in a Project Retrospective," *Journal of Systems and Software*, vol. 61, no. 3, pp. 173–187, 2002, ISSN: 0164-1212. DOI: 10.1016/S0164-1212(01)00146-7. [Online]. Available: https://doi.org/10.1016/S0164-1212(01)00146-7.

[63] S. Mohapatra and S. Mohanty, "Defect Prevention through Defect Prediction: A Case Study at Infosys," in *Proceedings of the IEEE International Conference on Software Maintenance. ICSM 2001*, IEEE, vol. 8, 2001, pp. 260–272.

[64] M. Saini and K. K. Chahal, "Change Profile Analysis of Open-source Software Systems to Understand Their Evolutionary Behavior," *Frontiers of Computer Science*, vol. 12, no. 6, pp. 1105–1124, 2018, ISSN:

2095-2228. DOI: 10.1007/s11704-016-6301-0. [Online]. Available: https://doi.org/10.1007/s11704-016-6301-0.

[65] ISDW Group and others, "1044-2009-IEEE Standard Classification for Software Anomalies," *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)*, pp. 1–23, 2010. DOI: 10.1109/IEEESTD.2010.5399061.

[66] S. Kumaresh and R. Baskaran, "Defect Prevention Based on 5 Dimensions of Defect Origin," *International Journal of Software Engineering & Applications (IJSEA)*, vol. 3, no. 4, pp. 87–98, 2012, ISSN: 0976-2221. DOI: 10.5121/ijsea.2012.3407.

[67] J. Rooney and L. Hauvel, "Root Cause Analysis For Beginners," *Quality Progress*, vol. 37, pp. 45–53, Jul. 2004.

[68] B. Andersen and T. Fagerhaug, *Root Cause Analysis, Second Edition: Simplified Tools and Techniques*. ASQ Quality Press, 2006, ISBN: 9780873896924. [Online]. Available: https://books.google.ru/books?id=i2S-nVO%5C_L-MC.

[69] S. Dalal and R. S. Chhillar, "Empirical Study of Root Cause Analysis of Software Failure," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 4, pp. 1–7, 2013, ISSN: 0163-5948. DOI: 10.1145/2492248.2492263. [Online]. Available: https://doi.org/10.1145/2492248.2492263.

[70] P. Jalote and D. Kamma, "Using Defect Analysis Feedback for Improving Quality and Productivity in Iterative Software Development," in *Proceeding of the 2005 International Conference on Information and Communication Technology*, IEEE, vol. 47, 2005, pp. 703–713. DOI: 10.1109/ITICT.2005.1609661.

[71] M. Anastassiu and G. Santos, "Resistance to Change in Software Process Improvement-an Investigation of Causes, Effects and Conducts," in *19th Brazilian Symposium on Software Quality*, ser. SBQS'20, São Luís, Brazil: Association for Computing Machinery, 2020, pp. 1–11, ISBN: 9781450389235. DOI: 10.1145/3439961.3439982. [Online]. Available: https://doi.org/10.1145/3439961.3439982.

[72] T. O. Lehtinen, J. Itkonen, and C. Lassenius, "Recurring Opinions or Productive Improvements—what Agile Teams Actually Discuss in Retrospectives," *Empirical Software Engineering*, vol. 22, no. 5, pp. 2409–2452, 2017, ISSN: 0036-8075.

[73] L. Chambers, "A Hazard Analysis of Human Factors in Safety-critical Systems Engineering," in *Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software - Volume 55*, ser. SCS '05, vol. 162, Sydney, Australia: Australian Computer Society, Inc., 2006, pp. 27–41.

[74] T. O. Lehtinen, R. Virtanen, V. T. Heikkilä, and J. Itkonen, "Why the Development Outcome Does Not Meet the Product Owners' Expectations?" In *Proceedings of the International Conference on Agile Software Development*, C. Lassenius, T. Dingsøyr, and M. Paasivaara, Eds., Cham: Springer International Publishing, 2015, pp. 93–104.

[75] R. G. Mays, C. L. Jones, G. J. Holloway, and D. P. Studinski, "Experiences with Defect Prevention," *IBM Systems Journal*, vol. 29, no. 1, pp. 4–32, 1990, ISSN: 0018-8670. DOI: 10.1147/sj.291.0004.

[76] S. K. Nayak, R. A. Khan, and M. R. Beg, "Reliable Requirement Specification: Defect Analysis Perspective," in *Proceedings of the Interna-*

tional Conference on Computing and Communication Systems, P. V. Krishna, M. R. Babu, and E. Ariwa, Eds., Springer Berlin Heidelberg, 2012, pp. 740–751.

[77] I. Saidani, A. Ouni, M. Chouchen, and M. W. Mkaouer, "Predicting Continuous Integration Build Failures Using Evolutionary Search," *Information and Software Technology*, vol. 128, p. 106 392, 2020, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2020.106392. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920301579.

[78] Q. Wang, N. Jiang, L. Gou, X. Liu, M. Li, and Y. Wang, "BSR: A Statistic-based Approach for Establishing and Refining Software Process Performance Baseline," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06, Shanghai, China: Association for Computing Machinery, 2006, pp. 585–594, ISBN: 1595933751. DOI: 10.1145/1134285.1134368. [Online]. Available: https://doi.org/10.1145/1134285.1134368.

[79] S. McConnell, "An Ounce of Prevention," *IEEE software*, vol. 18, no. 3, p. 5, 2001, ISSN: 0740-7459. DOI: 10.1109/MS.2001.922718.

[80] S. Kumaresh and R. Baskaran, "Experimental Design on Defect Analysis in Software Process Improvement," in *Proceedings of the 2012 International Conference on Recent Advances in Computing and Software Systems*, IEEE, 2012, pp. 293–298. DOI: 10.1109/RACSS.2012.6212683.

[81] CMMI Product Team, "CMMI for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1, Continuous Representation (CMMI-SE/SW/IPPD/SS, V1.1, Continuous)," Software Engineering Institute, Carnegie Mellon

University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2002-TR-011, 2002. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=6105.

[82] ——, “CMMI for Development, Version 1.2,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2006-TR-008, 2006. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8091.

[83] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, “Curating github for engineered software projects,” *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, 2017.

[84] H. Borges and M. T. Valente, “What’s in a github star? understanding repository starring practices in a social coding platform,” *Journal of Systems and Software*, vol. 146, pp. 112–129, 2018.

[85] M. O. F. Rokon, P. Yan, R. Islam, and M. Faloutsos, “Repo2vec: A comprehensive embedding approach for determining repository similarity,” in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2021, pp. 355–365.

[86] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue, “Mudablue: An automatic categorization system for open source repositories,” *Journal of Systems and Software*, vol. 79, no. 7, pp. 939–953, 2006.

[87] M. Jureczko and L. Madeyski, “Towards identifying software project clusters with regard to defect prediction,” in *Proceedings of the 6th international conference on predictive models in software engineering*, 2010, pp. 1–10.

[88] M. A. Saied, A. Ouni, H. Sahraoui, R. G. Kula, K. Inoue, and D. Lo, "Improving reusability of software libraries through usage pattern mining," *Journal of Systems and Software*, vol. 145, pp. 164–179, 2018.

[89] P. Pickerill, H. J. Jungen, M. Ochodek, M. Maćkowiak, and M. Staron, "Phantom: Curating github for engineered software projects using time-series clustering," *Empirical Software Engineering*, vol. 25, no. 4, pp. 2897–2929, 2020.

[90] D. Tsoukalas, M. Mathioudaki, M. Siavvas, D. Kehagias, and A. Chatzigeorgiou, "A clustering approach towards cross-project technical debt forecasting," *SN Computer Science*, vol. 2, no. 1, pp. 1–30, 2021.

[91] V. Lenarduzzi, N. Saarimäki, and D. Taibi, "The technical debt dataset," in *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*, 2019, pp. 2–11.

[92] Z. Xu, L. Li, M. Yan, *et al.*, "A comprehensive comparative study of clustering-based unsupervised defect prediction models," *Journal of Systems and Software*, vol. 172, p. 110 862, 2021.

[93] A. J. J. Tan, C. Y. Chong, and A. Aleti, "E-sc4r: Explaining software clustering for remodularisation," *Journal of Systems and Software*, vol. 186, p. 111 162, 2022.

[94] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[95] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[96] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *stat*, vol. 1050, p. 20, 2017.

[97] S. Brody, U. Alon, and E. Yahav, “How attentive are graph attention networks?” *arXiv preprint arXiv:2105.14491*, 2021.

[98] A. Barua, S. W. Thomas, and A. E. Hassan, “What are developers talking about? an analysis of topics and trends in stack overflow,” *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[99] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[100] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein, “Static test case prioritization using topic models,” *Empirical Software Engineering*, vol. 19, no. 1, pp. 182–212, 2014.

[101] C. Rosen and E. Shihab, “What are mobile developers asking about? a large scale study using stack overflow,” *Empirical Software Engineering*, vol. 21, no. 3, pp. 1192–1223, 2016.

[102] N. Klein, C. S. Corley, and N. A. Kraft, “New features for duplicate bug detection,” in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 324–327.

[103] P. Luangaram, W. Wongwachara, *et al.*, “More than words: A textual analysis of monetary policy communication,” *PIER Discussion Papers*, vol. 54, 2017.

[104] T. L. Griffiths and M. Steyvers, “Finding scientific topics,” *Proceedings of the National academy of Sciences*, vol. 101, no. suppl 1, pp. 5228–5235, 2004.

[105] A. Agrawal, W. Fu, and T. Menzies, “What is wrong with topic modeling? and how to fix it using search-based software engineering,” *Information and Software Technology*, vol. 98, pp. 74–88, 2018.

[106] M. Hoffman, F. Bach, and D. Blei, “Online learning for latent dirichlet allocation,” *advances in neural information processing systems*, vol. 23, 2010.

[107] C. Treude and M. Wagner, “Predicting good configurations for github and stack overflow topic models,” in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, IEEE, 2019, pp. 84–95.

[108] A. Strehl and J. Ghosh, “Cluster ensembles—a knowledge reuse framework for combining multiple partitions,” *Journal of machine learning research*, vol. 3, no. Dec, pp. 583–617, 2002.

[109] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, “Predicting issue types on github,” *Science of Computer Programming*, vol. 205, p. 102 598, 2021, ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2020.102598. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642320302069.

[110] S. Panichella, G. Canfora, and A. Di Sorbo, ““won’t we fix this issue?” qualitative characterization and automated identification of wontfix issues on github,” *Information and Software Technology*, vol. 139, p. 106 665, 2021.

[111] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu, “Exploring the characteristics of issue-related behaviors in github using visualization techniques,” *IEEE Access*, vol. 6, pp. 24 003–24 015, 2018.

[112] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Ticket tagger: Machine learning driven issue classification," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2019, pp. 406–409.

[113] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: Spectral clustering and normalized cuts," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 551–556.

[114] E. Schubert, S. Hess, and K. Morik, "The relationship of dbscan to matrix factorization and spectral clustering," in *LWDA*, 2018.

[115] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, 2002, pp. 849–856.

[116] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *kdd*, vol. 96, 1996, pp. 226–231.

[117] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

[118] R. A. Coelho, F. dos RN Guimaraes, and A. A. Esmin, "Applying swarm ensemble clustering technique for fault prediction using software metrics," in *2014 13th International Conference on Machine Learning and Applications*, IEEE, 2014, pp. 356–361.

[119] S. P. R. Puchala, J. K. Chhabra, and A. Rathee, "Ensemble clustering based approach for software architecture recovery," *International Journal of Information Technology*, pp. 1–7, 2022.

[120] J. Ghosh and A. Acharya, "Cluster ensembles," *Wiley interdisciplinary reviews: Data mining and knowledge discovery*, vol. 1, no. 4, pp. 305–315, 2011.

[121] S. Vega-Pons and J. Ruiz-Shulcloper, "A survey of clustering ensemble algorithms," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 25, no. 03, pp. 337–372, 2011.

[122] X. Z. Fern and C. E. Brodley, "Solving cluster ensemble problems by bipartite graph partitioning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 36.

[123] T. Li, C. Ding, and M. I. Jordan, "Solving consensus and semi-supervised clustering problems using nonnegative matrix factorization," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, IEEE, 2007, pp. 577–582.

[124] R. Dhanya, I. R. Paul, S. S. Akula, M. Sivakumar, and J. J. Nair, "F-test feature selection in stacking ensemble model for breast cancer prediction," *Procedia Computer Science*, vol. 171, pp. 1561–1570, 2020.

[125] E. Balas and M. J. Saltzman, "An algorithm for the three-index assignment problem," *Operations Research*, vol. 39, no. 1, pp. 150–161, 1991.

[126] H. P. Williams, "Integer programming," in *Logic and Integer Programming*, Springer, 2009, pp. 25–70.

[127] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[128] M. J. Saltzman, “Coin-or: An open-source library for optimization,” in *Programming languages and systems in computational economics and finance*, Springer, 2002, pp. 3–32.

[129] S. Mitchell, “An introduction to pulp for python programmers,” *Python Papers Monograph*, vol. 1, no. 14, p. 2009, 2009.

[130] S. A. Alkhodair, B. C. Fung, O. Rahman, and P. C. Hung, “Improving interpretations of topic modeling in microblogs,” *Journal of the Association for Information Science and Technology*, vol. 69, no. 4, pp. 528–540, 2018.

[131] C. Data61, *Stellargraph machine learning library*, https://github.com/stellargraph/stellargraph, 2018.

[132] Y. Bugayenko, K. Daniakin, M. Farina, *et al.*, *Metrics for recommending corrective and preventive actions (capas) in software development projects: A systematic literature review*, 2022. DOI: 10.21227/1zxb-q808. [Online]. Available: https://dx.doi.org/10.21227/1zxb-q808.

[133] D. M. Fernández, B. Penzenstadler, M. Kuhrmann, and M. Broy, “A Meta Model for Artefact-orientation: Fundamentals and Lessons Learned in Requirements Engineering,” in *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Part II*, D. C. Petriu, N. Rouquette, and Ø. Haugen, Eds., ser. MODELS’10, Oslo, Norway: Springer Berlin Heidelberg, 2010, pp. 183–197.

[134] D. Galin, “Software Process Quality Metrics, Concepts and practice,” in *Software Quality: Concepts and Practice*. John Wiley & Sons, Ltd, 2018, ISBN: 9781119134527. DOI: https://doi.org/10.1002/9781119134527.

ch21. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119134527.ch21. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119134527.ch21.

[135] K. H. Möller and D. J. Paulish, *Software Metrics. A Practitioner's Guide to Improved Product Development*, ser. Chapman & Hall computing. London: Chapman and Hall Computing; Piscataway, NJ. : IEEE Press, 1993, pp. 59–67, ISBN: 0412459000.

[136] International Organization for Standardization, *ISO Standard 9126: Software Engineering Product Quality, parts 1, 2 and 3*, 2003.

[137] M.-C. Lee, "Software Quality Factors and Software Quality Metrics to Enhance Software Quality Assurance," *British Journal of Applied Science & Technology*, vol. 4, no. 21, pp. 3069–3095, 2014, ISSN: 2231-0843.

[138] K. Daniakin, "Overcoming metric diversity in meta-analysis for software engineering: Proposed approach and a case study on its usage on the effects of software reuse," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1677–1679.

[139] Y. Singh and B. Goel, "A Step Towards Software Preventive Maintenance," *ACM SIGSOFT Software Engineering Notes*, vol. 32, no. 4, 10–es, 2007, ISSN: 0163-5948. DOI: 10.1145/1281421.1281432. [Online]. Available: https://doi.org/10.1145/1281421.1281432.

[140] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, *et al.*, "Constrained k-means clustering with background knowledge," in *Icml*, vol. 1, 2001, pp. 577–584.

[141] K. S. Hendis, "Quantifying Software Quality," in *Proceedings of the ACM '81 Conference*, ser. ACM '81, New York, NY, USA: Association for Computing Machinery, 1981, pp. 268–273, ISBN: 0897910494. DOI: 10.1145/800175.809892. [Online]. Available: https://doi.org/10.1145/800175.809892.

[142] G. Fox, "Performance Engineering As a Part of the Development Life Cycle for Large-scale Software Systems," in *Proceedings of the 11th International Conference on Software engineering*, ser. ICSE '89, Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1989, pp. 85–94. DOI: 10.1145/74587.74596. [Online]. Available: https://doi.org/10.1145/74587.74596.

[143] R. G. Mays, "Applications of Defect Prevention in Software Development," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 2, pp. 164–168, 1990, ISSN: 0733-8716. DOI: 10.1109/49.46867.

[144] B. W. Boehm, "Software Risk Management: Principles and Practices," *IEEE Software*, vol. 8, no. 1, pp. 32–41, 1991, ISSN: 0740-7459. DOI: 10.1109/52.62930.

[145] G. Travassos, F. Shull, M. Fredericks, and V. R. Basili, "Detecting Defects in Object-oriented Designs: Using Reading Techniques to Increase Software Quality," *ACM SIGPLAN Notices*, OOPSLA '99, vol. 34, no. 10, pp. 47–56, 1999. DOI: 10.1145/320384.320389. [Online]. Available: https://doi.org/10.1145/320384.320389.

[146] H. Younessi, "Managing Software Defects in an Object-oriented Environment," *Defense Software Engineering*, vol. 3, no. 1, pp. 13–16, 2003, ISSN: 1660-1769.

[147] L. Wallace, M. Keil, and A. Rai, "How Software Project Risk Affects Project Performance: An Investigation of the Dimensions of Risk and an Exploratory Model," *Decision sciences*, vol. 35, no. 2, pp. 289–321, 2004, ISSN: 0011-7315. DOI: https://doi.org/10.1111/j.00117315.2004.02059.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.00117315.2004.02059.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.00117315.2004.02059.x.

[148] X. Liu, G. Kane, and M. Bambroo, "An Intelligent Early Warning System for Software Quality Improvement and Project Management," *Journal of Systems and Software*, vol. 79, no. 11, pp. 1552–1564, 2006, Software Cybernetics, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2006.01.024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121206000239.

[149] A. S. White, "External Disturbance Control for Software Project Management," *International Journal of Project Management*, vol. 24, no. 2, pp. 127–135, 2006, ISSN: 0263-7863. DOI: https://doi.org/10.1016/j.ijproman.2005.07.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0263786305000852.

[150] Q. Song, M. Shepperd, M. Cartwright, and C. Mair, "Software Defect Association Mining and Defect Correction Effort Prediction," *IEEE Transactions on Software Engineering*, vol. 32, no. 2, pp. 69–82, 2006, ISSN: 0098-5589. DOI: 10.1109/TSE.2006.1599417.

[151] P. Björndal, K. Smiley, and P. Mohapatra, "Global Software Project Management: A Case Study," in *Proceedings of the 4th International Conference on Software Engineering Approaches for Offshore and Out-*

sourced Development, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 64–70. DOI: https://doi.org/10.1007/978-3-642-13784-6_7.

[152] C.-P. Chang, "Integrating Action-based Defect Prediction to Provide Recommendations for Defect Action Correction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 23, no. 02, pp. 147–172, 2013, ISSN: 0218-1940. DOI: 10.1142/S0218194013500022.

[153] S. Kumaresh and R. Baskaran, "Software Defect Prevention through Orthogonal Defect Classification (ODC)," *INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY*, vol. 11, no. 3, pp. 2393–2400, 2013, ISSN: 2277-3061. DOI: 10.24297/ijct.v11i3.1166. [Online]. Available: https://www.rajpub.com/index.php/ijct/article/view/1166ijct.

[154] S. Fatima, D. Mohd, R. Beg, and S. A. Siddiqui, "Improving Software Quality Using FMEA and FTA Defect Prevention Techniques in Design Phase," *International Journal of Computer Science and Information Technologies*, vol. 4, no. 1, pp. 178–182, 2013, ISSN: 0975-9646.

[155] M. Yan, Y. Fu, X. Zhang, D. Yang, L. Xu, and J. D. Kymer, "Automatically Classifying Software Changes Via Discriminative Topic Model: Supporting Multi-category and Cross-project," *Journal of Systems and Software*, vol. 113, pp. 296–308, 2016, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2015.12.019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412121500285X.

[156] W. Hu, J. C. Carver, V. Anu, G. Walia, and G. Bradshaw, "Defect Prevention in Requirements Using Human Error Information: An Empirical Study," in *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality*,

P. Grünbacher and A. Perini, Eds., Cham: Springer International Publishing, 2017, pp. 61–76, ISBN: 978-3-319-54045-0. DOI: 10.1007/978-3-319-54045-0_5.

[157] A. Freire, M. Perkusich, R. Saraiva, H. Almeida, and A. Perkusich, "A Bayesian Networks-based Approach to Assess and Improve the Teamwork Quality of Agile Teams," *Information and Software Technology*, vol. 100, pp. 119–132, 2018, ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2018.04.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584917300204.

[158] A. Mas, A.-L. Mesquida, and M. Pacheco, "Supporting the Deployment of Iso-based Project Management Processes with Agile Metrics," *Computer Standards & Interfaces*, vol. 70, p. 103 405, 2020, ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2019.103405. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920548919303575.

[159] G. Bhatia, *Keytotext*. [Online]. Available: https://github.com/gagan3012/keytotext.

[160] G. Dlamini, F. Jolha, Z. Kholmatova, and G. Succi, "Meta-analytical comparison of energy consumed by two sorting algorithms," *Information Sciences*, vol. 582, pp. 767–777, 2022, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2021.09.061. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025521009981.

[161] K. Daniakin, "Identifying" good" practices of developers using corrective and preventive actions and their impact on software metrics," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 2134, 2021, p. 012 013.

Appendix A

PRISMA 2020 Checklist

Table XVI: PRISMA 2020 Checklist. Template is taken from [36].

Section and Topic	Item #	Location
TITLE		
Title	1	1
ABSTRACT		
Abstract	2	2.1
INTRODUCTION		
Rationale	3	1
Objectives	4	1
METHODS		
Eligibility criteria	5	2.5
Information sources	6	2.4

Search strategy	7	2.3
Selection process	8	2.7
Data collection process	9	2.6
Data items	10a	2.8
	10b	2.8, 2.9
Study risk of bias assessment	11	5.3
Effect measure	12	-
Synthesis methods	13a	2.9
	13b	-
	13c	3.1
	13d	2.9
	13e	-
	13f	-
Reporting bias assessment	14	5.2
Certainty assessment	15	5.3
RESULTS		
Study selection	16a	2.6
	16b	2.6
Study characteristics	17	-
Risk of bias in studies	18	5
Results of individual studies	19	4

Results of syntheses	20a	4
	20b	-
	20c	-
	20d	-
Reporting biases	21	5.2
Certainty of evidence	22	5.3
DISCUSSION		
Discussion	23a	
	23b	5.1
	23c	5.1
	23d	4
OTHER INFORMATION		
Registration and protocol	24a	-
	24b	2.1
	24c	-
Support	25	-
Competing interests	26	7
Availability of data, code and other materials	27	8

Appendix B

Quality Scores

Table XVII: Scores - Quality Assessment.

Primary Study	Q1	Q2	Q3	Q4	Q5	Total	Rank
[141]	1	0.5	0	0	0	1.5	Poor
[41]	1	1	0	0.5	0	2.5	Good
[42]	1	0.5	0	0	0.5	2	Good
[142]	1	0.5	0	0	0.5	2	Good
[75]	1	1	1	0	1	4	Good
[43]	1	0.5	0	0	0.5	2	Good
[44]	0.5	1	0	0.5	0	2	Good
[143]	1	1	0	0	1	3	Good
[144]	1	0.5	0	0	0	1.5	Poor

[61]	1	1	1	1	1	5	Excellent
[23]	1	1	0	0.5	0	2.5	Good
[45]	1	1	1	1	0	4	Good
[145]	1	1	1	0	1	4	Good
[46]	1	1	0	0.5	0	2.5	Good
[63]	1	1	0.5	0	0	2.5	Good
[62]	1	1	1	1	0.5	4.5	Excellent
[146]	1	0.5	0	0	1	2.5	Good
[147]	0.5	1	0	0	0.5	2	Good
[70]	1	1	1	1	1	5	Excellent
[78]	1	1	1	1	1	5	Excellent
[73]	1	0.5	0	0	0.5	2	Good
[148]	1	1	0	0.5	0	2.5	Good
[149]	0.5	1	0	0	1	2.5	Good
[150]	1	1	0	1	0.5	3.5	Good
[60]	1	1	1	0.5	0.5	4	Good
[57]	1	0.5	0	0	0	1.5	Poor
[151]	0.5	0.5	0	1	0	2	Good
[47]	0.5	1	0	0.5	0	2	Good
[58]	1	1	1	1	1	5	Excellent
[48]	0.5	1	0	1	0	2.5	Good

[76]	1	0.5	0	0	1	2.5	Good
[80]	1	1	1	0.5	1	4.5	Excellent
[66]	1	1	1	1	1	5	Excellent
[69]	1	0.5	0	0	1	2.5	Good
[55]	1	1	0	1	0.5	3.5	Good
[152]	0.5	1	1	1	0	3.5	Good
[153]	1	1	0	0	1	3	Good
[154]	1	0.5	0	0	1	2.5	Good
[10]	0.5	0	0	0.5	1	2	Good
[53]	1	0.5	0	0	1	2.5	Good
[54]	1	0.5	0	0	1	2.5	Good
[74]	1	1	0.5	0	1	3.5	Good
[155]	0.5	1	0	0	0	1.5	Poor
[39]	1	1	0	1	0	3	Good
[72]	1	1	1	0	0	3	Good
[52]	0.5	1	0	1	0	2.5	Good
[156]	1	1	1	0.5	1	4.5	Excellent
[157]	1	1	1	0	0.5	3.5	Good
[49]	1	1	0	0	1	3	Good
[64]	0.5	1	0	1	0	2.5	Good
[59]	1	0.5	0	0	0	1.5	Poor

[50]	0.5	1	0	1	0	2.5	Good
[38]	0.5	0.5	0	0	0	1	Poor
[71]	0.5	0.5	0	0	1	2	Good
[158]	0.5	0.5	0	0	0	1	Poor
[77]	0.5	1	0	1	0	2.5	Good
[56]	1	1	0	1	0.5	3.5	Good
[40]	1	1	0	1	0	3	Good

Appendix C

Statistical Data Obtained from the Reading Log

In Table XVIII, the column "Samples" represents the number of data samples obtained from the relevant study. The columns "Metrics", "Anomalies", and "Actions" represent the number of data samples corresponding to the relevant column. The column "Metrics-Anomalies" displays the number of connections among these data types. The same applies to the other two columns: "Anomalies-Actions" and "Metrics-Actions" and to the column "Metrics-Anomalies-Actions". The row "Total" displays the sum of the values for the relevant column, whereas the row "Count" shows the number of unique data samples that are gathered from the studies.

Table XVIII: Statistical information concerning the data gathered from the reading log.

Study	Samples	Metrics	Anomalies	Actions	Metrics-Anomalies	Anomalies-Actions	Metrics-Actions	Metrics-Anomalies-Actions
[141]	9	2	7					
[41]	14	4		10			1	
[42]	46	29	4	17		17		
[142]	4	1	4	4	1	4	1	1
[75]	18		4	18		18		
[43]	20	15	1	5		1		
[44]	4	4	1		4			
[143]	2		2	2		2		
[144]	10		10	10		10		
[61]	9	9	7	7	9	9	9	9
[23]	5	2		5			2	

[45]	10	6	3	7	7	7	7	7
[145]	5		5	1		5		
[46]	19	19	1	1	1	1	1	1
[63]	18	4	17	18	4	18	4	4
[62]	8	3	5	8	1	5	3	1
[146]	41		11	39		22		
[147]	78		78					
[70]	7	1	1	7	1	1	1	1
[78]	4	3	4	1	3	3	3	3
[73]	30		16	25		30		
[148]	7	7	3		3			
[149]	40			40				
[150]	9	3	6					
[60]	9	5	6	4	4	4		
[57]	8	4	8	6	8	7	7	7
[151]	1	1						
[47]	35	35						
[58]	14	1	5	14	14	14	14	14
[48]	13	13	1		13			
[76]	28		20	13		28		
[80]	40	2	22	39	2	40	2	2

[66]	30	1	30	30	30	30	30	30
[69]	9	5	7	1	7	1		
[55]	7	7	1	1	3	3	3	3
[152]	10	10						
[153]	21		12	21		21		
[154]	4		4	4		4		
[10]	20		20					
[53]	4	3	4	4	3	4	3	3
[54]	21	11	10	10	20	21	20	20
[74]	29		9	16		29		
[155]	1	1						
[39]	18	18	1		18			
[72]	34		5	34		34		
[52]	13	13						
[156]	12	12	6	1	6	6	6	6
[157]	17	17	17		17			
[49]	25	18	3		25			
[64]	5	5						
[59]	140	80		47			133	
[50]	11	11						
[71]	30		1	29		29		

[158]	7	7						
[77]	33	33	1		33			
[56]	5	4	3		3			
[40]	6	6	1		6			
Total	1077	435	387	499	246	428	250	112
Count	1063	384	374	494	233	409	250	112

Appendix D

Attributes of Collected Data

This appendix describes the attributes related to the data gathered from the study we included in our final reading log. Specifically, we collected 3 types of relevant data. These are metrics, anomalies, and actions. In order to capture the relevance and significance of the data for our research, we also specified several attributes for each data type. In addition, we used these attributes to assess the quality of the studies we included in this SLR. The attributes we collected are as follows:

Table XIX: Data attributes.

Data Attribute	Description
Study	Each primary study is represented by its ID.
Title	Study title.
Year of publication	The year in which the study was published.
Authors	How many people participated in the study.

Raw metrics	The raw metric characterising the study.
Metrics	The metric/s under which we grouped similar raw metrics. This information can be found in the data file attached to this SLR.
Metrics formulas	The mathematical formula describing the metric.
Metric descriptions	The definition and description of the metric used.
Metric notes	Additional information about the symbols used in the formula or terms used in the description.
Possible metric values	The range of values that the metric can have or the range of metric measurements.
Metric context	The contextual usage of the metric in the study.
Is the metric used for detecting Anomalies?	"Yes", whether the metric is used for detecting anomalies, otherwise it is "No". We note here that some papers declared that specific metrics have been used for detecting anomalies, but these anomalies have not been identified. We set "Yes" for this kind of data samples but next the attribute "Anomaly" has not been set.
Anomalies	The anomaly obtained from the textual analysis of the study included in the reading log.
Anomaly descriptions	Additional descriptive information about the anomaly in question.
Root causes	The root causes of the encountered anomaly/ies or of those triggered by actions.
Metric threshold	The value of the metric or its category (low or high), which triggered the relevant anomaly/action.
Is the anomaly handled?	"Yes", if the obtained anomaly is fixed, otherwise it is "No". We note here that some papers declared that specific anomalies have been fixed, but the corrective actions have not been identified in the study. We set "Yes" in this attribute for this kind of data samples but the next attribute "Suggested Actions" has not been set.

Suggested actions	The action gathered from the study included in the reading log.
Action categories	This attribute specifies the type of the action obtained. The possible values for this attribute are 'Corrective', 'Preventive' and 'Enhancement'. 'Corrective' actions represent the actions applied to fix or correct the anomaly, while 'Preventive' actions are used to avoid the recurrence of the anomalies in the future. 'Enhancement' actions are actions adopted to increase software quality.
Action sources	This attribute captures the origin or source of the action obtained. The possible values for this attribute are "Experimental", "Observational" and "Industrial". "Observational" actions originated from authors' observations, "Experimental" actions were developed based on experiments conducted in labs whereas "Industrial" actions were suggested based on experiments conducted by companies or in an industrial context.
Handling approaches	The approaches that are suggested and/or used for fixing the anomalies encountered.
Used ML methods	The machine learning algorithms used for detecting the anomalies.
Presence of action impact	"Yes", if the actions' impact was measured and reported in the study, otherwise it was "No".
Action impact metrics	The metric/s used for measuring actions' impact. This captures how much improvement is perceived after applying the desired action.
Percentages of actions' impact	The relative change obtained in the measurements of metrics' impact before and after implementing the suggested action.

Appendix E

Software Metrics Used in This Work

Table XX: GitHub API metrics used in this study.

Number	Metric Name	Metric Description
1	[Commits] Average additions	Average number of lines added through all the commits
2	[Commits] Average deletions	Average number of lines deleted through all the commits
3	[Commits] Average files changed	Average number of files changed through all the commits
4	[Commits] Average message length (chars)	Average length of messages within a commit (in chars)
5	[Commits] Count	Number of commits to this repository

6	[Commits] Days since first	Number of days from the first commit
7	[Commits] Days since last	Number of days from the last commit
8	[Commits] Maximum per day	Maximum number of commits per day
9	[Commits] Per day (True)	Average number of commits (counted only days with commits)
10	[Commits] Total lines added	Total number of lines added through all the commits
11	[Commits] Total lines deleted	Total number of lines deleted through all the commits
12	[Contributors Top-100] Average additions	Average number of lines added through all the commits by top-100 contributors
13	[Contributors Top-100] Average commits	Average number of commits made by top-100 contributors
14	[Contributors Top-100] Average deletions	Average number of lines deleted through all the commits by top-100 contributors
15	[Contributors Top-100] Average participation weeks	Number of participation weeks of top-100 contributors
16	[Contributors] Count	Number of contributors
17	[Forks] Count	Number of forks
18	[Forks] Max per day	Maximum number of forks per day
19	[Issues] Average body len (chars)	Average time to close an issue
20	[Issues] Average comment len (chars)	Average length of issue message (in chars)
21	[Issues] Average comments	Average number of comments
22	[Issues] Average labels	Average number of labels in issues

23	[Issues] Average title len (chars)	Average length of issue title (in chars)
24	[Issues] Count	Number of issues
25	[Issues] Labels	Number of issue labels
26	[Issues] Maximum per day	Maximum number of issues per day
27	[Issues] Open	Number of open issues
28	[Issues] Per day	Average number of issues per day (counted all days)
29	[Issues] Per day (True)	Average number of issues (counted only days with issues)
30	[Issues] Total comments	Total number of issue comments
31	[Pulls] Average body len (chars)	Average length of a body within a pull (in chars)
32	[Pulls] Average comments	Average number of comments in a pull
33	[Pulls] Average commits	Average number of commits in a pull
34	[Pulls] Average files changed	Average number of files changed through all the pulls
35	[Pulls] Average labels	Average number of labels within a pull
36	[Pulls] Average lines added	Average number of lines deleted through all pulls
37	[Pulls] Average lines deleted	Average number of lines added through all pulls
38	[Pulls] Average review comments	Average number of review comments in a pull
39	[Pulls] Average title len (chars)	Average length of a title within a pull (in chars)
40	[Pulls] Count	Number of pulls
41	[Pulls] Created per day (True)	Average number of created pull per day (counted only days with pulls)

42	[Pulls] Maximum created per day	Maximum number of created pulls per day
43	[Pulls] Total lines added	Total number of lines added through all pulls
44	[Pulls] Total lines deleted	Total number of lines deleted through all pulls
45	[Releases] Average asset downloads	Average number of asset downloads
46	[Releases] Average asset size	Average asset size
47	[Releases] Average assets	Average number of assets
48	[Releases] Average body len (chars)	Average length of a body within a release (in chars)
49	[Releases] Average title len (chars)	Average length of a title within a release (in chars)
50	[Releases] Count	Number of releases
51	[Releases] Tags	Number of tags
52	[Releases] Total downloads	Total number of downloads
53	[Repo] Age (days)	Age of a repository (in days)
54	[Repo] Branches	Number of branches
55	[Repo] Deployments	Number of deployments
56	[Repo] Milestones	Number of milestones
57	[Repo] Network members	Number of network members
58	[Repo] Programming Languages	Number of programming languages used in a repo
59	[Repo] Readme length (chars)	Number of readme length (in chars)
60	[Repo] Size	Size of repository
61	[Repo] Topics	Number topics of repository
62	[Repo] Watchers	Number of watchers
63	[Repo] Workflows	Number of workflows

64	[Stars] Count	Number of stars
65	[Stars] Maximum per day	Maximum number of stars per day
66	[Stars] Per day (True)	Average number of stars
67	[Workflow Runs] Average duration (ms)	Average duration of workflow runs (ms)
68	[Workflow Runs] Average fails per day	Average number of failed workflow runs per day (counted all days)
69	[Workflow Runs] Average failure duration (ms)	Average duration of failure workflow runs (ms)
70	[Workflow Runs] Average success duration (ms)	Average duration of success workflow runs (ms)
71	[Workflow Runs] Average successes per day (True)	Average number of success workflow runs per day (counted days with workflows)
72	[Workflow Runs] Count	Number of workflow runs

Appendix F

The Clustering Results for Each of the Repository Sets

Table XXI: Optimal values of clustering algorithms' parameters according to different validity indices across repository sets.

Clustering algorithm	Silhouette	Calinski-Harabasz	Davies-Bouldin
	Set 1		
DBSCAN ε=2.25	0.363	6.882	1.366
DBSCAN ε=1.25	0.119	12.58	2.624
K-means k=2	0.238	27.769	1.718
K-means k=7	0.19	14.602	1.542
Spectral k=2	0.233	26.563	1.736
Spectral k=10	0.148	4.554	0.907
	Set 2		

DBSCAN ε=2.35	0.374	4.96	1.36
DBSCAN ε=1.3	0.134	12.144	3.212
K-means k=2	0.219	28.331	1.74
K-means k=10	0.135	9.058	1.687
Spectral k=2	0.219	28.199	1.743
	Set 3		
DBSCAN ε=2.2	0.401	4.173	0.458
DBSCAN ε=1.1	0.134	11.332	2.682
K-means k=2	0.262	25.73	1.684
K-means k=3	0.269	19.615	1.493
K-means k=9	0.15	10.699	1.385
Spectral k=2	0.328	5.851	0.822
Spectral k=3	0.249	15.519	1.471
	Set 4		
DBSCAN ε=2.2	0.355	5.357	1.661
DBSCAN ε=1.25	0.146	11.88	2.951
K-means k=2	0.248	27.942	1.697
K-means k=10	0.098	9.006	1.469
Spectral k=2	0.248	27.942	1.697
Spectral k=3	0.25	3.463	1.088
	Set 5		
DBSCAN ε=2.35	0.366	3.804	2.12
DBSCAN ε=2.1	0.37	6.777	2.462
DBSCAN ε=1.25	0.177	11.188	3.187
K-means k=2	0.261	25.131	1.714
K-means k=8	0.133	9.516	1.251

Spectral k=2	0.262	24.775	1.715
Spectral k=8	0.114	8.426	1.524
	Set 6		
DBSCAN ε=2.2	0.434	6.984	1.163
DBSCAN ε=1.3	0.133	12.331	3.084
K-means k=2	0.228	25.094	1.803
K-means k=7	0.203	12.798	1.293
Spectral k=2	0.228	25.094	1.803
Spectral k=9	0.09	7.168	1.5
	Set 7		
DBSCAN ε=2.25	0.39	4.596	2.007
DBSCAN ε=1.95	0.372	5.822	1.982
DBSCAN ε=1.1	0.054	7.479	2.905
K-means k=2	0.216	22.79	1.902
K-means k=8	0.127	10.515	1.513
Spectral k=2	0.216	22.79	1.902
Spectral k=6	0.14	6.81	1.404
	Set 8		
DBSCAN ε=2.2	0.414	6.183	2.03
DBSCAN ε=1.25	0.195	13.758	2.972
DBSCAN ε=0.95	0.084	11.183	1.808
K-means k=2	0.279	32.629	1.542
K-means k=9	0.161	13.796	1.216
Spectral k=2	0.279	32.113	1.545
Spectral k=6	0.189	13.62	1.459
	Set 9		

DBSCAN ε=2.25	0.37	3.102	1.902
DBSCAN ε=1.15	0.128	10.458	3.101
K-means k=2	0.21	23.182	1.835
K-means k=9	0.143	9.49	1.162
Spectral k=2	0.21	23.182	1.835
Spectral k=6	0.188	8.942	1.22
		Set 10	
DBSCAN ε=2.2	0.45	8.885	1.4
DBSCAN ε=2.0	0.394	12.258	1.596
K-means k=2	0.258	25.677	1.783
K-means k=10	0.101	9.814	1.472
Spectral k=2	0.258	25.677	1.783
Spectral k=5	0.208	13.384	1.596

Table XXII: Values of objective function on two clusters for different consensus clustering algorithms. The highest value of the objective function is underlined for each of the repository sets.

Consensus clustering algorithm	Objective function: Average normalized mutual information									
	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8	Set 9	Set 10
Hypergraph-partitioning algorithm	0.612	$\underline{1.281}$	0.519	0.371	0.442	0.881	0.421	0.807	0.963	$\underline{0.822}$
Meta-clustering algorithm	0.261	0.704	0.583	0.347	0.17	0.685	0.475	$\underline{1.002}$	0.746	0.563
Hybrid bipartite graph formulation	$\underline{0.786}$	0.898	0.645	0.415	0.545	0.633	0.865	0.548	0.574	0.6
Cluster-based similarity partitioning algorithm	0.465	0.704	0.358	0.404	0.463	$\underline{0.923}$	0.418	0.534	$\underline{1.04}$	0.325
Nonnegative matrix factorization	0.622	0.704	$\underline{0.648}$	$\underline{0.604}$	$\underline{0.61}$	0.685	$\underline{0.954}$	1.002	0.746	0.563

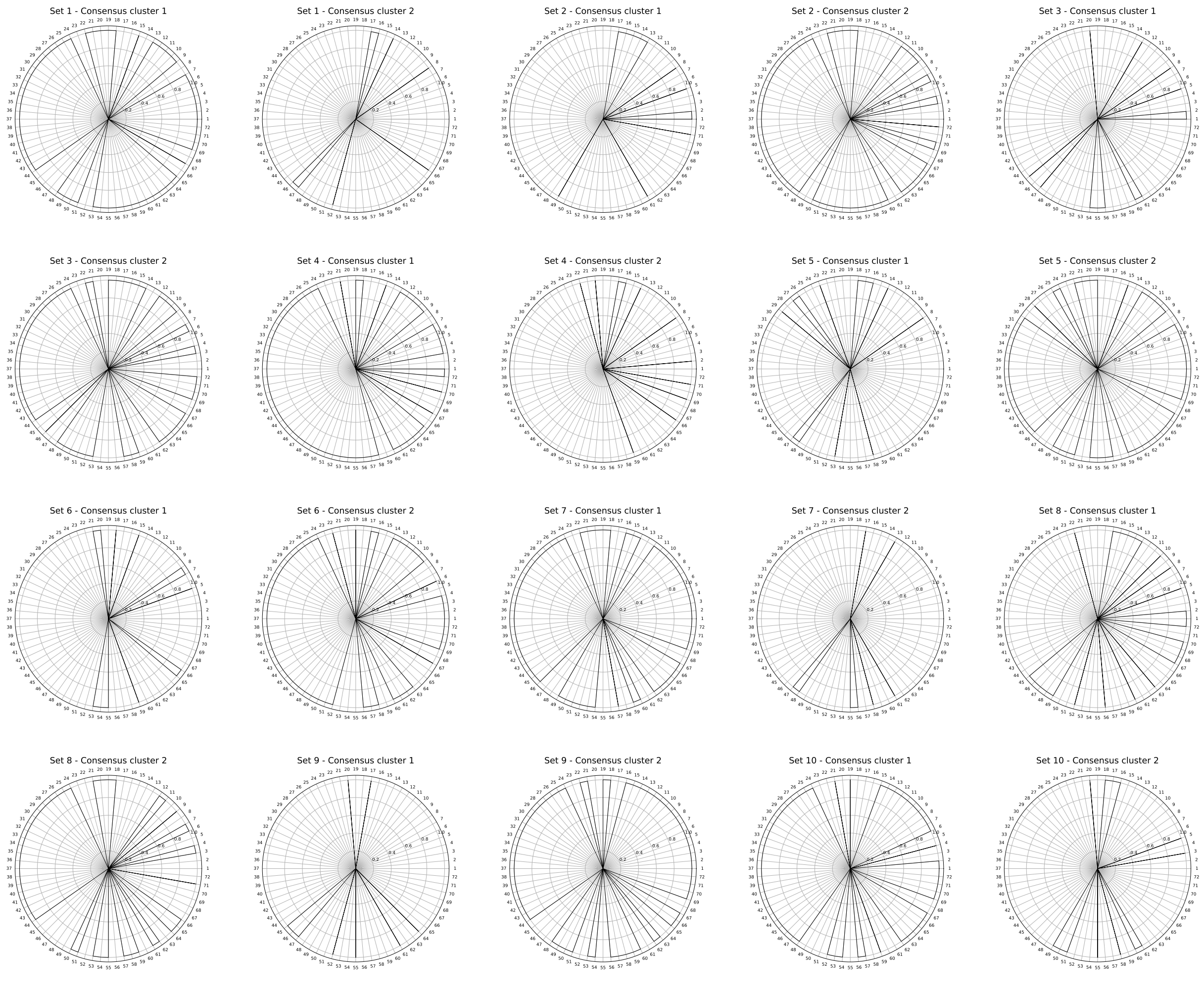


Figure F.1: Consensus cluster prototypes for each repository set.

Appendix G

Topic Modeling Results

Here we present the results we obtained from topic modeling on issues and commits. The tables in this chapter show the topics represented by the top words and the number of words for each topic in the topic model. The labels of the topics are generated from the top words using the open source tool "keytotext" [159]. This tool uses pre-trained transformers to generate sentences from the input keywords and it is usually used for topic labeling and fine tuning the outputs of topic modeling but the drawback of this tool is that it is trained only on non-technical text whereas in our case we are dealing with technical text. In order to improve the meaningfulness of the generated labels, we manually paraphrased them but it is better to have an automatic way of labeling the technical text and indeed this method needs a lot of labeled technical text which is out of our concentration in this thesis.

Table XXIII: Topics extracted from “fixing” commits with corresponding 5-top words.

#	Commit topic	Word count per topic
1	['remov', 'test', 'file', 'non', 'api']	108
2	['renam', 'instead', 'provid', 'failur', 'ignor']	68
3	['resolv', 'issu', 'thi', 'work', 'delet']	111
4	['chang', 'sort', 'function', 'packet', 'read']	86
5	['flag', 'us', 'function', 'user', 'vip']	70
6	['return', 'object', 'address', 'maximum', 'superus']	55
7	['test', 'case', 'regress', 'npe', 'move']	74
8	['use', 'string', 'json', 'instead', 'version']	100
9	['display', 'format', 'featur', 'onli', 'version']	95
10	['debug', 'prefer', 'root', 'asdf', 'wifi_ssid']	57
11	['element', 'text', 'html', 'creat', 'document']	84
12	['connect', 'reset', 'client', 'cannot', 'respons']	85
13	['setstr', 'meshtast', 'run', 'configur', 'wifi']	69
14	['result', 'wrap', 'strategi', 'around', 'empti']	69
15	['onli', 'thi', 'user', 'allow', 'end']	162
16	['option', 'compil', 'resolv', 'execut', 'madskristensen']	59
17	['type', 'temperatur', 'field', 'rang', 'content']	85
18	['thank', 'transact', 'properti', 'object', 'error']	79
19	['chang', 'alarm', 'mode', 'class', 'see']	85
20	['handl', 'data', 'forc', 'includ', 'correct']	101
21	['structur', 'devic', 'serial', 'first', 'creat']	76
22	['task', 'claus', 'bf', 'queue', 'main']	42
23	['support', 'name', 'event', 'call', 'function']	98
24	['support', 'search', 'index', 'implement', 'dot']	73
25	['request', 'merg', 'pull', 'defin', 'synonym']	63
26	['messag', 'improv', 'client', 'text', 'bodi']	64
27	['check', 'code', 'error', 'miss', 'null']	119
28	['properli', 'valu', 'valid', 'select', 'disabl']	121
29	['bug', 'get', 'fail', 'charact', 'incorrect']	91
30	['make', 'use', 'order', 'doc', 'thi']	132
31	['issu', 'depend', 'script', 'replac', 'jar']	98
32	['updat', 'link', 'readm', 'version', 'instal']	92

Table XXIV: Commit topic labels generated from top 10 words and manually paraphrased.

#	Commit topic label
1	Remove test file and use different API
2	Fix the failure of a package by renaming or changing the ID
3	Resolve issue related to the work report
4	Change sort function and test the new API
5	When using meta queries, you need to take the size of the tables into consideration.
6	Update a configuration parameter to activate some command
7	Perform a regression test
8	Use a different json version
9	Refactor the display format of the app
10	Debug Reset button in the app
11	Use null-safe variables
12	Handle client connection to the server
13	Configure wifi module
14	Handle the authentication strategy with the client
15	Support online users
16	Add option to resolve the app build
17	Fix the content field in mgrid in python
18	Update local transaction
19	Change build mode of the project
20	Handle issue related to updating the data
21	Create specific volume for the device
22	Update task dialog
23	Set position:fixed for fullscreen mode
24	Implement support for misplaced dot on input
25	Uninstall the new installed package
26	Improve client notification
27	Check the error code of the null output
28	Properly add select validation to the form
29	Prevent the incorrect character number in the window of the android app
30	Fix ordering of list items
31	Fix script dependencies
32	Update readme file

Table XXV: Topics extracted from “bug” issues with corresponding 5-top words.

#	Issue topic	Word count per topic
1	['animation', 'route', 'page', 'app', 'src']	5937
2	['server', 'error', 'client', 'player', 'reproduce']	6767
3	['component', 'entity', 'gree', 'homeassistant', 'py']	4223
4	['search', 'name', 'query', 'str', 'result']	4862
5	['mongodb', 'connect', 'kafka', 'org', 'converter']	3205
6	['time', 'start', 'service', 'second', 'stop']	8979
7	['dll', 'php', 'address', 'vendor', 'thread']	4130
8	['lib', 'module', 'node', 'ghost', 'logger']	4857
9	['data', 'model', 'train', 'py', 'input']	4712
10	['java', 'com', 'lang', 'run', 'util']	10499
11	['map', 'rest', 'metadata', 'row', 'swagger']	2862
12	['file', 'line', 'py', 'lib', 'self']	15103
13	['lua', 'framexml', 'bagnon', 'interface', 'component']	4152
14	['behavior', 'reproduce', 'expected', 'bug', 'step']	8294
15	['jar', 'xml', 'scala', 'user', 'play']	4316
16	['npm', 'err', 'node', 'http', 'react']	4469
17	['none', 'highlight', 'logback', 'development', 'exe']	3215
18	['build', 'cpp', 'lib', 'library', 'src']	6593
19	['string', 'type', 'data', 'json', 'value']	11148
20	['hie', 'bios', 'ghc', 'haskell', 'cabal']	2903
21	['angular', 'cli', 'ember', 'mocha', 'bower']	2630
22	['html', 'template', 'class', 'href', 'div']	5925
23	['module', 'node', 'lib', 'user', 'webpack']	7942
24	['product', 'subscription', 'cart', 'order', 'price']	5285
25	['airflow', 'docker', 'info', 'compose', 'postgresql']	4437
26	['link', 'page', 'menu', 'click', 'browser']	9247
27	['file', 'directory', 'root', 'rw', 'user']	4544
28	['go', 'transaction', 'git', 'github', 'com']	3131
29	['flutter', 'src', 'dart', 'org', 'springframework']	3029
30	['this', 'bug', 'product', 'jet', 'widget']	8392
31	['test', 'download', 'py', 'error', 'exception']	3532
32	['lua', 'function', 'defined', 'bagnon', 'addons']	6146
33	['date', 'value', 'status', 'end', 'summary']	3228
34	['io', 'client', 'netty', 'vertx', 'connection']	4624
35	['table', 'mysql', 'id', 'db', 'data']	4603
36	['item', 'bag', 'bank', 'bagnon', 'character']	10712
37	['log', 'numjobs', 'iodepth', 'randread', 'iop']	7070
38	['php', 'woocommerce', 'wp', 'plugins', 'index']	7162
39	['public', 'new', 'void', 'string', 'import']	7590
40	['run', 'install', 'command', 'build', 'sh']	10533
41	['java', 'org', 'junit', 'hoverfly', 'engine']	3891
42	['openid', 'app', 'cluster', 'exporter', 'google']	4776
43	['this', 'issue', 'work', 'problem', 'working']	57062
44	['this', 'set', 'value', 'result', 'using']	25246
45	['request', 'response', 'error', 'url', 'api']	12986
46	['atom', 'app', 'package', 'remote', 'edit']	6747
47	['user', 'email', 'password', 'account', 'permission']	7112
48	['field', 'form', 'value', 'post', 'type']	9412
49	['import', 'python', 'py', 'module', 'file']	7068
50	['system', 'microsoft', 'window', 'runtime', 'aspnetcore']	6470
51	['meshtastic', 'serial', 'debug', 'gpio', 'arduino']	5322
52	['android', 'view', 'script', 'com', 'app']	5155
53	['error', 'context', 'addon', 'running', 'software']	6479
54	['active', 'language', 'inactive', 'syntax', 'autocomplete']	3240
55	['div', 'class', 'de', 'md', 'col']	5272
56	['version', 'docker', 'issue', 'false', 'information']	8347
57	['version', 'latest', 'package', 'dependency', 'update']	9705
58	['image', 'color', 'style', 'text', 'font']	6266
59	['filter', 'listing', 'post', 'page', 'grid']	10507
60	['java', 'samczsun', 'sun', 'net', 'ssl']	4183
61	['error', 'this', 'file', 'get', 'any']	27735
62	['function', 'error', 'this', 'console', 'undefined']	11047
63	['rb', 'gem', 'ruby', 'lib', 'redmine']	3383
64	['embedded', 'swimmingseadragon', 'bagnon', 'dev', 'sanctimoniousswamprat']	8110

Table XXVI: Issue topic labels generated from top 10 words and manually paraphrased.

#	Issue topic label
1	issue in the animation image in the web page.
2	bug in the version of the client player app
3	bug in climate component of the home assistant app
4	bug in search result of the query
5	issue in connection to mongodb
6	service crashes due to memory issues
7	issue in php deployment using magento
8	issue in logger module of node.js app
9	issue in input/output layer of the model during training
10	bug in concurrent thread of java app
11	issue in column/row display in swagger typescript rest api
12	issue in one of the python libraries
13	error in bagnon addon in lua component
14	unexpected behaviour of the app
15	bug in the the player written in scala
16	error in the installation of the npm modules react and node-gyp
17	error in gui development in aragon
18	unknown error in library while building a cpp app
19	issue in data type of json key/value content
20	build issue of haskell project
21	issue in installing the ember-mocha module of npm.
22	issue in html template.
23	issue in one of the react modules the in node.js app
24	issue in payment page of woocommerce app
25	bug in docker compose file related to postgresql
26	issue in open button of the page
27	illegal usage of the directory group permission
28	bug in transaction of the website's cpanel
29	issue in one bean of the flutter app
30	filter problem in woobuilder plugin
31	failure in integration with travis ci
32	issue in one of the lua addons
33	error in summary description of the app
34	client connect issue to the database
35	issue in backup of the database
36	issue in one of the addons
37	errors in configuration parameters in json files
38	plugin issue in woocommerece app
39	issue in access modifiers of a variable
40	docker build failure
41	issues in testing the app with a testing module
42	issue in database exporter
43	some unspecified issue
44	bug in setting values of some fields
45	error in request/response of the api client related to access token
46	issue in one of addons of atom editor
47	access permission bug in user login windows
48	bug in form fields while doing post
49	bug in one of the python libraries related to sharepoint services
50	object exception in aspnet app
51	bug in access port in arduino app
52	issue in using ajax in android app
53	error in one of the addons of the software patch
54	issue in themes of the app
55	bugs in html elements of the webpage
56	issue in docker version used
57	deprecation issue in one of the dependencies
58	bugs in css files
59	bugs in grid widgets used in the webpage
60	security issues in the app
61	file errors
62	type error in the console
63	issue in one of the ruby packages
64	issue in lua addons used for embedded modules

Appendix H

Our Publications and Submissions

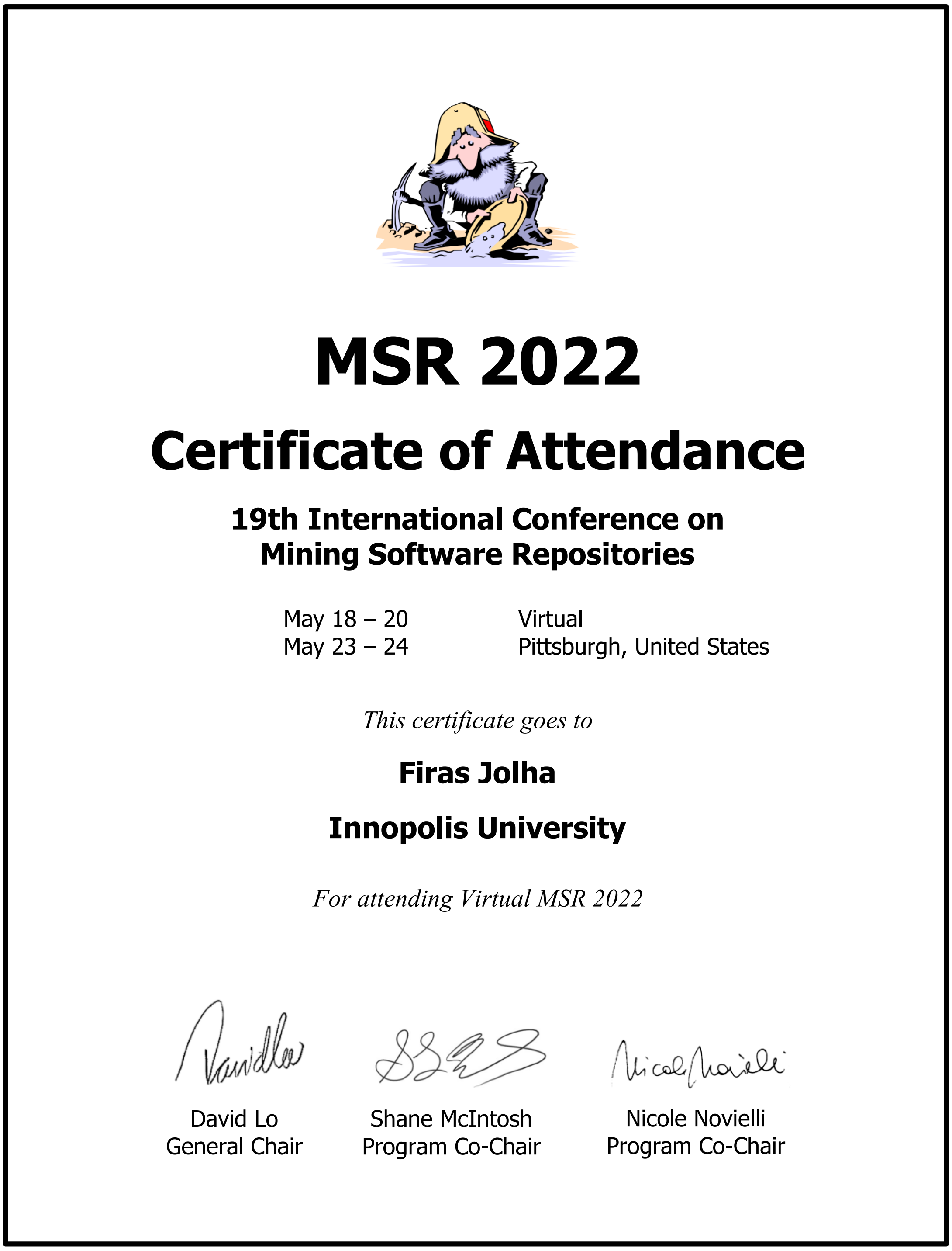
- We submitted the paper Extracting corrective actions from code repositories to the industrial track of the conference MSR 2022 and it has been **accepted**. However, it is not yet published as of date of writing this thesis. Firas Jolha was the presenter of the paper on 20 May 2022 and the participation certificate is shown in H.1.

- We submitted the systematic literature review "Metrics for Recommending Corrective and Preventive Actions (CAPAs) in Software Development Projects: a Systematic Literature Review" to Transactions on Software Engineering.

- We uploaded and published the extracted data [132] from the SLR papers in IEEE-DataPort.

- Firas Jolha et al. published a Q1 paper entitled "Meta-analytical comparison of energy consumed by two sorting algorithms" [160] to the Journal of Information Sciences. He recently got a citation on the work.

- Kirill Daniakin published the paper on identifying “good” practices of developers using corrective and preventive actions and their impact on software metrics to the Journal of Physics [161].
- Kirill Daniakin published and presented the paper entitled “Overcoming metric diversity in meta-analysis for software engineering: proposed approach and a case study on its usage on the effects of software reuse” [138] that deals with the diversity of metrics used by software engineering researchers – the discovery that the SLR confirmed – to one of the leading software engineering conferences – ESEC/FSE. The paper has more than 70 downloads in less than one year as per the official publisher site.
- Kirill Daniakin participated in the ACM Student Research Competition (SRC) held at ESEC/FSE 2021. The participation certificate is presented in H.2.



MSR 2022

Certificate of Attendance

19th International Conference on
Mining Software Repositories

May 18 – 20	Virtual
May 23 – 24	Pittsburgh, United States

This certificate goes to

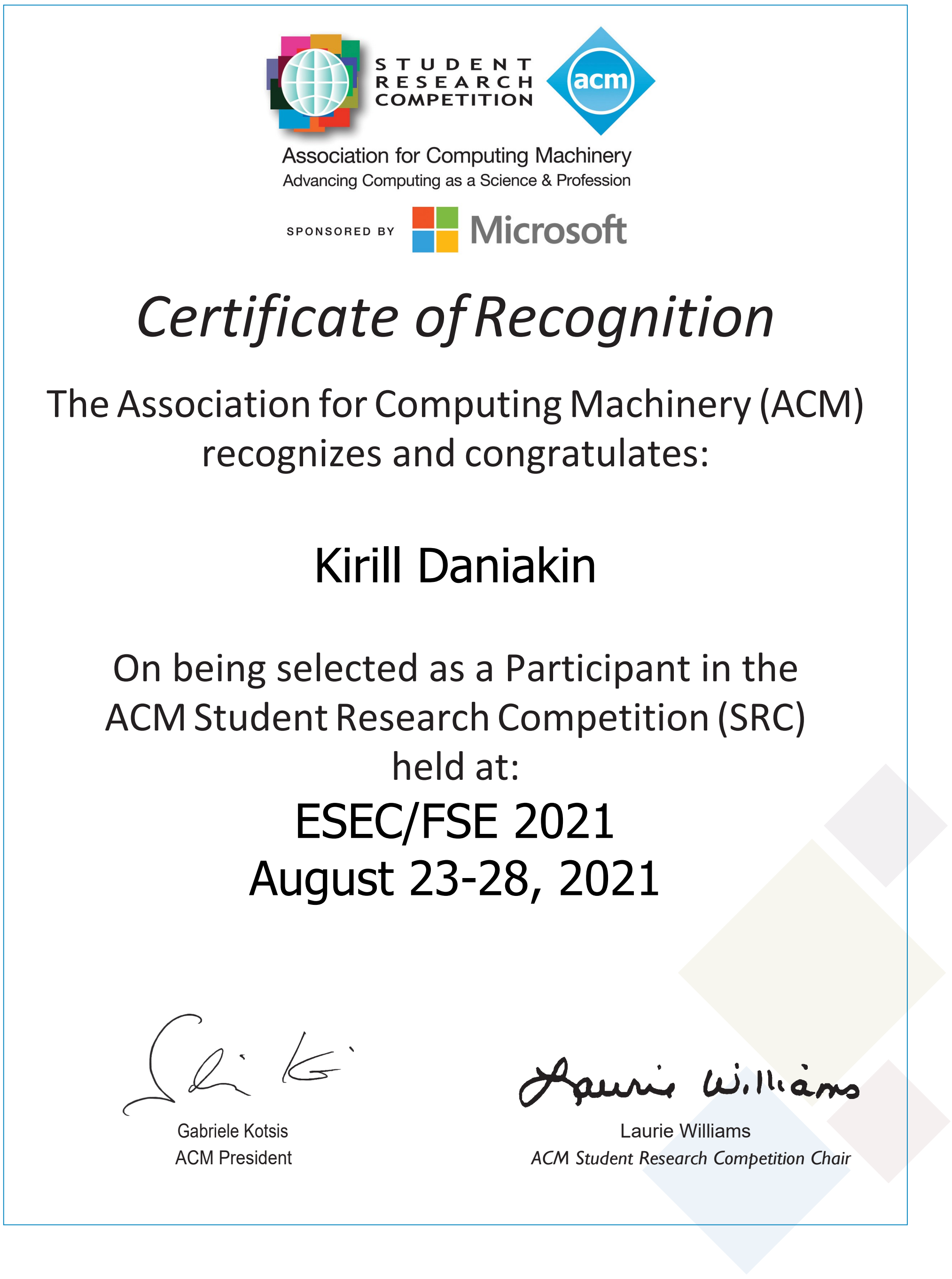
Firas Jolha

Innopolis University

For attending Virtual MSR 2022

[illegible]	[illegible]	[illegible]
David Lo General Chair	Shane McIntosh Program Co-Chair	Nicole Novielli Program Co-Chair

Figure H.1: Participation certificate at MSR'22 Conference.



STUDENT RESEARCH COMPETITION

acm

Association for Computing Machinery
Advancing Computing as a Science & Profession

SPONSORED BY Microsoft

Certificate of Recognition

The Association for Computing Machinery (ACM)
recognizes and congratulates:

Kirill Daniakin

On being selected as a Participant in the
ACM Student Research Competition (SRC)
held at:

ESEC/FSE 2021
August 23-28, 2021

[illegible]

Gabriele Kotsis
ACM President

Laurie Williams

Laurie Williams
ACM Student Research Competition Chair

Figure H.2: Participation certificate in the ACM Student Research Competition held at ESEC/FSE 2021.