2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion) | 979-8-3503-2263-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE-COMPANION58688.2023.00051

Detecting Scattered and Tangled Quality Concerns in Source Code to Aid Maintenance and Evolution Tasks

Rrezarta Krasniqi
Dept. of Computer Science and Engineering
University of North Texas
Denton, TX, USA
Rrezarta.Krasniqi@unt.edu

***Abstract*—Quality concerns, such as reliability, security, usability concerns, among others, are typically well-defined and prioritized at the requirement level with the set goal of achieving high quality, robust, user-friendly, and trustworthy systems. However, quality concerns are challenging to address at the implementation level. Often they are scattered across multiple modules in the codebase. In other instances, they are tangled with functional ones within a single module. Reasoning about quality concerns and their interactions with functional ones while being hindered by the effects of scattered and tangled code can only yield to more unseen problems. For example, developers can inadvertently retrofit new bugs or wrongly implement new features that deviate from original system requirement specifications. The goal of this thesis is twofold. First, we aim to detect quality concerns implemented at code level to differentiate them from functional ones when they are scattered across the codebase. Second, we aim to untangle quality concerns from unrelated changes to gain a detailed knowledge about the history of specific quality changes. This knowledge is crucial to support consistency between the requirements-and-design and to verify architecture conformance. From the practical stance, developers could gain a breadth of understanding about quality concerns and their relations with other artifacts. Thus, with more confidence, they could perform code modifications, improve module traceability, and provide a better holistic assessment of change impact analysis.**

***Index Terms*—Quality Concerns, Quality Bugs, Tangled Quality Concerns, Scattered Quality Concerns, Software Maintenance**

I. Introduction

Addressing quality concerns such as reliability, performance, security concerns are essential for ensuring that software systems remain safe and stable at all times. However, when maintenance issues arise, developers spend immense intellectual effort to detect them, understand their relation with other artifacts to prevent the potential of retrofitting new bugs. Two key factors why software systems become difficult to maintain and evolve is that implemented quality become unfavorably constrained by the effects of the *tangled code* [11], [14], [15] and the *scattered code* [10], [26], [30]. Tangled code happens when several concerns with defined responsibilities become inter-tangled within a single module [7].

As a result, these constraints impede developers predominately from understanding the functionalities that such models convey. This situation indirectly affects software maintainability, software reuse, and the cost of code changes [3]. Scattered code happens when snippets of code that convey a certain intent or concern become disintegrated across multiple modules. One way to address this issue is to adopt requirement specification templates that describe scattered concerns. However, this approach is impracticable primarily because software constantly evolves which creates the risk of these pre-defined templates to become outdated. Therefore, a more holistic and cohesive description of the knowledge about the implemented quality concerns can help software developers understand quality features better, their relation with other functional features. Hence, developers would with more confidence perform correct modifications and enhancements for new version releases. The underpinning knowledge about the implemented quality features undoubtedly maximizes developer's productivity in effort spent on existing tasks.

In the completed portion of this thesis, we incorporated a series of approaches for detecting scattered quality concerns across various software artifacts. Initially, we built a feature-based selection method where we detected six types of quality concerns in bug report short descriptions for efficient bug triaging [21]. Then, we built BugReportSoftQualDetector [22] where we employed a feature extraction method. Specifically, we leveraged a weighted combination of *semantics*, *lexical*, and *shallow* features using the Random Forest ensemble model. The *semantics* feature was further augmented by instantiating a BERT transformation model [8]. We then introduced SoftQualTopicDetector [31], that could cluster scattered quality concerns across various artifacts into a meaningful topic hierarchy to infer a set of candidate classes relevant for recommending repair of quality bugs. Finally, we introduced an NLP-based framework SoftQualDetector [23] that focused on detecting quality classes scattered across the codebase. SoftQualDetector additionally extracted short keyword summaries from the detected quality-related classes in the context of *semantics*, *keyword importance*, and *textual* characteristics of the source code.

As part of ongoing research effort, now we are working on a novel technique that aims to untangle quality concerns from source code histories to aid maintenance and evolution tasks. By splitting quality tangled changes into a set of untangled ones, we can more accurately predict future quality bugs, monitor quality concern changes for traceability and auditing related tasks, prioritize them based on their vulnerability level such as security and/or privacy concerns, and obtain insights about their dependencies with other software artifacts.

979-8-3503-2263-7/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE-Companion58688.2023.00051

II. Related Work

Software concerns are unavoidable [18]. Quality concerns are no exception either [19]. Most closely related work has focused on detecting quality concerns from an architectural ground [1], [2]. The work of Gopalakrishnan *et al.* [12] introduced a recommendation system to predict missing quality concerns present in various classes. A recent work by Márquez and Astudillo [24] proposed a new strategy for identifying availability quality concerns to support security architectural design suitable for microservice-based systems. A work by Harty [13] discussed developers' involvement with usability bugs. Zaman *et al.* [34] studied performance bugs and included specific guidelines on how to go by detecting and fixing them. The work of Jin *et al.* [17] examined 109 performance bugs using efficiency rules to detect them. While the efforts of the researchers are undoubtedly novel, they exclude the analysis of quality concerns at large. In our line of research, we followed the taxonomy of ISO 25010 standard [27] and studies several quality attributes. We also stress the importance of detecting quality concerns from the semantic context and feature-based context that past studies have partially addressed.

III. Research Design and Evaluation

A. Research Methodology

We followed the well-known goal-question-metric (GQM) paradigm [4] to answer our research questions. We focused on three characteristics: *goals*, *quality focus*, and *perspective*.

• The thesis **goal** is to demonstrate how effective and efficient our approaches are in detecting quality concerns for different purposes of software development.

• The thesis **quality focus** is on building several transformative frameworks that support the detection of quality concerns for i) bug triaging, ii) bug repair, iii) source code comprehension, and iv) source code maintenance and evolution—using various machine learning, information retrieval, and advanced topic modeling techniques including other intermediary techniques such as structural and textual analysis for code transformation.

• The research **perspective** is both of the:

(i) *researchers*, that want to evaluate and/or assess our approaches at different levels of software development.

(ii) *software developers* that want to adopt our approaches to handle quality concerns during software development.

B. Research Questions

In the early stages of the software development, software architects and developers make critical decisions which quality concerns must be satisfied and implemented. However, as software evolves and maintenance increases, quality features tend to erode. Thus, they become scattered and tangled across the codebase. Under these effects, quality concerns can become difficult to detect, understand, and maintain. Subsequently, they indirectly affect users' experiences at large. In this thesis, we aim to close these gaps and explore several RQs.

RQ1: How well does our three-pronged feature selection model performs for detecting quality-related information present in bug report short descriptions?

RQ2: How well does our three-pronged feature extraction approach (that is contextualized in terms of semantic, lexical, and shallow features) perform for enhancing prediction of quality concerns in bug report short descriptions?

RQ3: To what extent can our proposed approach accurately detect quality concerns scattered across the codebase to infer bug-fixing candidate classes?

RQ4: To what extent our proposed approach can detect and extract informed keyword summaries to comprehend scattered quality concerns across the codebase?

RQ5: How do quality-related changes at source code level impact software sustainability during maintenance tasks?

RQ6: What are common challenges that emanate when handling quality concerns during software development?

RQ7: How well can our novel approach untangle quality concerns to aid maintenance and evolution tasks?

C. Evaluation Metric

To assess the accuracy of [RQ1–RQ4], we employed standard information retrieval metrics of precision and recall. Precision (i.e., the fraction of retrieved quality-related files that are correct). Recall (i.e., the fraction of correct quality-related files that are retrieved). We also computed F1 score. F1 computes the harmonic mean of precision and recall.

D. Quality Concerns Taxonomy

To establish quality concerns taxonomy, we used both the FURPS quality model [9] and ISO 25010 standard [27]. The FURPS quality model [9] categorizes quality-related concerns into five types, such as functionality, usability, reliability, performance, and supportability. The ISO 25010 standard categorizes quality concerns into eight types: functional suitability, efficiency, compatibility, usability, reliability, security, maintainability, and portability. In our study, we focused on selecting six most common ones derived both from ISO 25010 standard [27] and the FURPS quality model [9]. Our premise on making such selection is because they are most common quality attributes that software developers focus on [35].

E. Objects of Analysis

We used six diverse OSS software systems from the JIRA issue tracker as objects of analysis [29]. Derby is a relation database implemented in Java. Drools is a business logic integration platform (BLiP). Groovy is an open-source optionally typed and dynamic language. Maven is a management and comprehension tool. Pig is an analytic platform, comprising of infrastructure for evaluating high-level languages. Infinispan is an in-memory data grid that provides deployment mechanisms capabilities for storing, managing, and processing data.

TABLE I
RESULTS OF DETECTED QUALITY-RELATED BUG REPORTS RETURNED BY FOUR DISTINCT CONFIGURATIONS

	TF-IDF			**TF-IDF+χ^2**			TF-IDF+MI			TF-IDF+ET		
ML Classification Algorithms	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Support Vector Machine (SVM)	0.48	0.27	0.23	0.71	0.64	0.64	0.52	0.39	0.40	0.55	0.66	0.55
Logistic Regression (LR)	0.51	0.27	0.22	0.78	0.64	0.66	0.51	0.39	0.38	0.44	0.64	0.50
Random Forest (RF)	0.47	0.27	0.24	**0.76**	**0.70**	**0.70**	0.53	0.43	0.43	0.68	0.68	0.58
Multinomial Naive Bayes (MNB)	0.50	0.27	0.23	0.69	0.58	0.55	0.38	0.38	0.69	0.41	0.64	0.50
kNeighbors Classifier (kNN)	0.48	0.27	0.23	0.66	0.41	0.43	0.62	0.33	0.40	0.51	0.65	0.54

TABLE II
PERFORMANCE RESULTS OF COMBINED APPROACHES AND LDA APPROACH [16]

Approach	Cmb(TF, TR)			Cmb(TF, Y)			Cmb(TR, Y)			**All(TF, TR, Y)**			LDA [16]		
Project	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Derby	0.73	0.77	0.73	0.66	0.71	0.65	0.69	0.74	0.69	0.78	0.80	0.78	0.76	0.81	0.76
Drools	0.83	0.85	0.82	0.77	0.83	0.73	0.81	0.83	0.81	0.88	0.91	0.88	0.65	0.70	0.67
Groovy	0.82	0.85	0.83	0.84	0.87	0.86	0.79	0.89	0.86	0.88	0.92	0.90	0.71	0.75	0.73
Maven	0.87	0.89	0.87	0.80	0.92	0.80	0.84	0.85	0.83	0.90	0.94	0.88	0.69	0.72	0.68
Infinispan	0.76	0.77	0.72	0.71	0.73	0.71	0.70	0.74	0.70	0.85	0.89	0.86	0.72	0.74	0.73
Pig	0.85	0.83	0.83	0.74	0.76	0.74	0.85	0.88	0.86	0.90	0.94	0.92	0.68	0.74	0.72
Averages	0.81	0.83	0.80	0.75	0.80	0.75	0.78	0.82	0.79	**0.86**	**0.90**	**0.87**	0.70	0.74	0.72

TABLE III
COMPARISON OF OUR APPROACH (SOFTQUALTOPICDETECTOR) AGAINST THE BASELINE [28] AND THE SOTA [12] ON DETECTING SIX QUALITY TYPES

Quality Type	Baseline [28]			SOTA [12]			**Our Approach [31]**		
	P	R	F1	P	R	F1	P	R	F1
Reliability	0.74	0.85	0.80	0.81	0.82	0.82	0.98	0.96	0.97
Maintainability	0.51	0.71	0.61	0.72	0.64	0.68	0.89	0.87	0.88
Usability	0.60	0.54	0.57	0.46	0.36	0.41	0.67	0.63	0.65
Portability	0.27	0.16	0.22	0.82	0.81	0.82	0.87	0.80	0.84
Security	0.67	0.29	0.48	0.60	0.52	0.56	0.70	0.68	0.70
Performance	1.00	0.91	0.90	0.61	0.68	0.65	0.95	0.95	0.95
Averages	0.63	0.58	0.60	0.67	0.64	0.65	**0.84**	**0.82**	**0.83**

TABLE IV
OUR APPROACH [22] WITH BOOTSTRAPPING-VALIDATION

Quality Type	TP	FP	TN	FN	P	R	F1
Reliability	874	2	24	1	0.89	0.96	0.93
Maintainability	891	0	9	1	0.98	0.98	0.98
Usability	807	9	78	6	0.88	0.93	0.90
Portability	839	10	39	4	0.86	0.94	0.90
Security	884	1	15	0	0.84	0.97	0.90
Performance	855	4	39	3	0.89	0.88	0.89
Other (FR or NC)	716	12	152	21	0.94	0.95	0.95
Averages					**0.90**	**0.94**	**0.92**

IV. COMPLETED WORK

A. RQ1: Feature-based Detection of Quality Concerns

By answering our RQ1, we automatically detected quality-related concerns that most likely would cause defects [21]. We built an ML-based classifier that uses various feature selection models to detect six emerging quality types as defined by ISO 25010 [27] and the FURPS quality model [9]. We reported the results in Table I The results, showed that the use of TF-IDF+χ^2 features in combination with Random Forest model, outperformed other approaches and achieved P=0.76, R=0.70, and F1=0.70 to detect quality concerns in bug reports.

B. RQ2: Semantic-based Detection of Quality Concerns

To answer RQ2, initially we implemented a multi-model framework referred to as BUGREPORTSOFTQUALDETECTOR [20] where we leveraged a weighted combination of *semantics*, *lexical*, and *shallow* features using the *Random Forest* model. Then, we enhanced BUGREPORTSOFTQUALDETECTOR [22] where our original semantic-based algorithm was further augmented by instantiating a BERT transformation model [8]. We reported the results of BUGREPORTSOFTQUALDETECTOR [22] in Table IV which indicated that the extraction of semantic context from bug report along with the boosting mechanism of BERT model and the use of bootstrapping-validation produced much favorable results compared to our original work [20].

C. RQ3: Hierarchical Representation of Quality Concerns

To answer RQ3, we implemented a lightweight framework (SOFTQUALTOPICDETECTOR) [31], that leverages a Hierarchical Dirichlet Process (HDP) [33] topic modeling technique along with other supporting IR and ML techniques, including integration of structural and textual analyses techniques to capture hierarchical topical relationships among quality features that potentially yield to detection of quality bugs. Our approach can cluster scattered quality concerns into a meaningful hierarchy to infer a set of candidate classes relevant for recommending repair of quality bugs. We have compared our results with the most closely related work that of Poshyvanyk *et al.* [28] and Gopalakrishnan *et al.* [12]. As reported in Table III, our approach returned relatively higher P=0.84, R=0.82, and F1=0.83 than its counterpart approaches.

D. RQ4: Holistic Comprehension of Quality Concerns

To answer RQ4, we introduced a lightweight framework (SOFTQUALDETECTOR) [23], that combines three unsuper-

TABLE V
PUBLISHED WORK RESULTING FROM THIS THESIS SEPARATED BY WORK IN PROGRESS OR PLANNED

Ref.	Title/Topic	Venue	ˆRank	RQs
[21]	Automatically Capturing Quality-Related Concerns in Bug Report Descriptions for Efficient Bug Triaging	EASE'22	A	RQ1
[22]	A Multi-Model Framework for Semantically Enhancing Detection of Quality-Related Bug Report Descriptions	EMSE'23	Q1	RQ2
[31]	A Hierarchical Topical Modeling Approach for Recommending Repair of Quality Bugs	SANER'23	A	RQ3
[23]	Towards Semantically Enhanced Detection of Emerging Quality-Related Concerns in Source Code	SQJO'23	Q1	RQ4
–	A Task-Driven Approach to Guide Integration of Quality Concerns into a Unified Team Vocabulary	EASE'23	A	RQ5
–	Challenges and Perceptions in Exploring Quality Concerns during Software Development	EASE'23	A	RQ6
–	Quality Concerns Untangled: Generating Context-Aware Rationales from Source Code Change History	**TBD	–	RQ7

ˆ Metrics obtained as of 2023 via CORE Conference (http://www.core.edu.au/conference-portal) and Scimago Rankings (https://scimagojr.com/)

vised techniques such as TextRank (TR) [25], Yake (Y) [6], and TF-IDF (TF) [32] for generated a rich-set of quality-related code summaries. SOFTQUALDETECTOR also provides a 3D rich visualization mechanism to monitor quality concerns across the codebase so that developers can easily observe the relation of quality concerns with various classes and packages. Our evaluation of $1,248$ manually annotated Java classes showed that SOFTQUALDETECTOR outperformed several baseline approaches and state-of-the-art approaches. As depicted in Table II, The reported results showed that our approach All(TF,TR,Y) not only outperformed the combinations of approaches such as those derived from paired combinations such as Cmb(TF,TR), Cmb(TF,Y), and Cmb(TR,Y) but also the LDA variant employed by Hindle *et al.* [16]

V. ONGOING AND FUTURE WORK

A. Integration of Quality Concerns during Maintenance Tasks

RQ5: THE CASE STUDY: In our recent work, we performed a case study that required upper-level graduate students to perform a series of development tasks individually and on team bases. Specifically, we collected teams' feedback in relation to integration of quality concerns by providing three various tasks related to (1) perfective maintenance (i.e., adding a new feature), (2) corrective maintenance (i.e., fixing a bug), and (3) preventive maintenance (i.e., code refactoring). The results revealed that when *corrective maintenance* activities took place, the number of changed classes fluctuated (i.e., sometimes increased, sometimes decreased). Also, the results revealed that when *preventive maintenance* took place, the number of classes required fewer changes (i.e., decreased). In contrast, when *perfective maintenance* took place, the number of classes required major changes (i.e., increased substantially).

B. Challenges of Quality Concerns during Maintenance Tasks

RQ6: THE SURVEY STUDY: we then performed a survey study comprising 58 open-ended and closed-ended questions. We focused on collecting the feedback from the upper-level graduate students when they performed maintenance tasks asked in RQ5. After conducting a mixed quantitative and qualitative studies, the top-3 predominant challenges that participants faced indicated that quality concerns were 1) time consuming to go over code and other sources, 2) time consuming due to their entanglement with functional requirements, and 3) overwhelming to deal with because of their scattered nature across several artifacts and external APIs.

C. Untangling Quality Concerns from Code Change History

RQ7: TOOL BUILDING: The feedback that we received from both the case study (RQ5) and the survey study (RQ6) stress the emerging need to introduce a suitable tool that would be capable of untangling quality-related code changes from functional ones. We argue that underpinning knowledge about implemented quality concerns is essential for the cost-effective maintenance when predicting bugs and for co-evolution of large-scale software systems. Hence, by answering RQ7, we aim to separate parts of a history of code changes relevant to quality concerns. Our conjecture is that the tangled changes which are often bundled are problematic because it makes review reversion of tasks difficult and historical analyses of the data less reliable. Our high-level approach will consider several steps. First, we will analyze co-evolution across pairs of artifacts and identify quality-related features (e.g., security features) to determine in what context they occurred. Then, we need to explore the possibility of untangling code across multiple pairs of artifacts and perform impact change analysis to explain the context of quality changes across sequential versions of software systems.

EXPERIMENTAL CONTEXT. Once our tool is fully implemented, next we will perform a case study with industrial participants to evaluate its effectiveness in an industrial environment. We plan to carry the case study in three ways. First, we will conduct a feedback session with 10–16 experienced developers. We will then inquire them to use our tool to evaluate the history of untangled quality changes. Finally, we will ask participants feedback based on a set of user experience questions derived from the SU usability scale by Brooke [5].

VI. CONTRIBUTION AND PROPOSED TIMELINE

In 2022 and 2023, two conference publications were accepted at International Conference on Evaluation and Assessment in Software Engineering (EASE'22) and IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER'23) (see Table V). We also received recent acceptances of two journal articles. Those are Software Quality (SQJO'23) and Empirical Software Engineering (EMSE'23) with the latter one receiving a special invitation from the (SANER'21 [21]). We also conducted a survey study (RQ5, EASE'23) and a case study (RQ6, EASE'23). Currently, both full track papers are 'in review.' The ultimate goal is to finalize PhD thesis in May 2024 with the completion of tool building that we briefly introduced to answer RQ7.

REFERENCES

[1] David Ameller, Claudia Ayala, Jordi Cabot, and Xavier Franch. Non-functional Requirements in Architectural Decision Making. *IEEE software*, 30(2):61–67, 2012.

[2] David Ameller, Xavier Burgués, Dolors Costal, Carles Farré, and Xavier Franch. Non-functional Requirements in Model-Driven Development of Service-Oriented Architectures. *Science of Computer Programming*, 168:18–37, 2018.

[3] Elisa LA Baniassad, Gail C Murphy, Christa Schwanninger, and Michael Kircher. Managing Crosscutting Concerns during Software Evolution Tasks: An Inquisitive Study. In *1st International Conference on Aspect-Oriented Software Development*, pages 120–126, 2002.

[4] Victor R Basili. Goal Question Metric Paradigm. *Encyclopedia of Software Engineering*, pages 528–532, 1994.

[5] John Brooke et al. SUS-A Quick and Dirty Usability Scale. *Usability Evaluation in Industry*, 189(194):4–7, 1996.

[6] Ricardo Campos, Vítor Mangaravite, Arian Pasquali, Alípio Jorge, Célia Nunes, and Adam Jatowt. YAKE! Keyword Extraction from Single Documents using Multiple Local Features. *Information Sciences*, 509:257–289, 2020.

[7] Gerardo Canfora, Luigi Cerulo, and Massimiliano Di Penta. On the Use of Line Co-change for Identifying Crosscutting Concern Code. In *2006 22nd IEEE International Conference on Software Maintenance*, pages 213–222. IEEE, 2006.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding. *ArXiv preprint. ArXiv:1810.04805*, 2018.

[9] R. Geoff Dromey. A Model for Software Product Quality. *IEEE Transaction in Software Engineering*, 21(2):146–162, 1995.

[10] Marc Eaddy, Thomas Zimmermann, Kaitlin D Sherwood, Vibhav Garg, Gail C Murphy, Nachiappan Nagappan, and Alfred V Aho. Do Crosscutting Concerns Cause Defects? *IEEE transactions on Software Engineering*, 34(4):497–515, 2008.

[11] Stephen G Eick, Todd L Graves, Alan F Karr, J Steve Marron, and Audris Mockus. Does Code Decay? Assessing the Evidence from Change Management Data. *IEEE Transactions on Software Engineering*, 27(1):1–12, 2001.

[12] Raghuram Gopalakrishnan, Palak Sharma, Mehdi Mirakhorli, and Matthias Galster. Can latent topics in source code predict missing architectural tactics? In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 15–26. IEEE, 2017.

[13] Julian Harty. Finding Usability Bugs with Automated Tests. *Communications of the ACM*, 54(2):44–49, 2011.

[14] Kim Herzig, Sascha Just, and Andreas Zeller. The Impact of Tangled Code Changes on Defect Prediction Models. *Empirical Software Engineering*, 21(2):303–336, 2016.

[15] Kim Herzig and Andreas Zeller. The Impact of Tangled Code Changes. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 121–130. IEEE, 2013.

[16] Abram Hindle, Neil A Ernst, Michael W Godfrey, and John Mylopoulos. Automated Topic Naming to Support Cross-Project Analysis of Software Maintenance Activities. In *8th Working Conference on Mining Software Repositories*, pages 163–172. ACM, 2011.

[17] Guoliang Jin, Linhai Song, Xiaoming Shi, Joel Scherpelz, and Shan Lu. Understanding and Detecting Real-World Performance Bugs. In *ACM Conference on Programming Language Design and Implementation*, pages 77–88, 2012.

[18] Rrezarta Krasniqi. Extractive Summarization of Related Bug-fixing Comments in Support of Bug Repair. In *IEEE/ACM International Workshop on Automated Program Repair*, pages 31–32. IEEE, 2021.

[19] Rrezarta Krasniqi. Recommending Bug-fixing Comments from Issue Tracking Discussions in Support of Bug Repair. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 812–823. IEEE, 2021.

[20] Rrezarta Krasniqi and Ankit Agrawal. Analyzing and Detecting Emerging Quality-Related Concerns across OSS Defect Report Summaries. In *Proceedings of the 28th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 12–23. IEEE, 2021.

[21] Rrezarta Krasniqi and Hyunsook Do. Automatically Capturing Quality-Related Concerns in Bug Report Descriptions for Efficient Bug Triaging. In *Proceedings of the 23rd International Conference on Evaluation and Assessment in Software Engineering (EASE) 2022*, pages 10—-19. ACM, 2022.

[22] Rrezarta Krasniqi and Hyunsook Do. A multi-model framework for semantically enhancing detection of quality-related bug report descriptions. *Empirical Software Engineering*, 28(2):1–62, 2023.

[23] Rrezarta Krasniqi and Hyunsook Do. Towards semantically enhanced detection of emerging quality-related concerns in source code. *Software Quality Journal*, pages 1–51, 2023.

[24] Gastón Márquez and Hernán Astudillo. Identifying Availability Tactics to Support Security Architectural Design of Microservice-Based Systems. In *Proceedings of the 13th European Conference on Software Architecture-Volume 2*, pages 123–129, 2019.

[25] Rada Mihalcea and Paul Tarau. TextRank: Bringing Order into Text. In *Conference on Empirical Methods in Natural Language Processing*, 2004.

[26] Ana Moreira, João Araújo, and Isabel Brito. Crosscutting Quality Attributes for Requirements Engineering. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, pages 167–174, 2002.

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

[28] Denys Poshyvanyk, Yann-Gaël Guéhéneuc, Andrian Marcus, Giuliano Antoniol, and Vaclav Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Trans on Software Engineering*, 33:420–432, 2007.

[29] Michael Rath, David Lo, and Patrick Mäder. Analyzing Requirements and Traceability Information to Improve Bug Localization. In *Proceedings of the 15th International Conference on Mining Software Repositories (MSR)*, pages 442–453, 2018.

[30] Martin P Robillard and Gail C Murphy. Representing Concerns in Source Code. *ACM Transactions on Software Engineering & Methodology*, 16(1):3, 2007.

[31] Rrezarta Krasniqi and Hyunsook Do. A Hierarchical Topical Modeling Approach for Recommending Repair of Quality Bugs. In *Proceeding of the 30th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 12–23. IEEE, 2023.

[32] Gerard Salton and Christopher Buckley. Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[33] Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Hierarchical Dirichlet Processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006.

[34] Shahed Zaman, Bram Adams, and Ahmed E. Hassan. A Qualitative Study on Performance Bugs. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 199–208. IEEE, 2012.

[35] Jie Zou, Ling Xu, Weikang Guo, Meng Yan, Dan Yang, and Xiaohong Zhang. Which Non-Functional Requirements Do Developers Focus On? An Empirical Study on Stack Overflow Using Topic Analysis. In *12th IEEE International Conference on Mining Software Repositories (MSR)*, pages 446–449. IEEE, 2015.