Refactoring-Aware Code Review: A Systematic Mapping Study

Flavia Coelho
Systems and Computing Department
Federal University of Campina Grande
Campina Grande, Brazil
flavia@copin.ufcg.edu.br

Tiago Massoni
Systems and Computing Department
Federal University of Campina Grande
Campina Grande, Brazil
massoni@computacao.ufcg.edu.br

Everton L. G. Alves
Systems and Computing Department
Federal University of Campina Grande
Campina Grande, Brazil
everton@computacao.ufcg.edu.br

***Abstract*—Modern Code Review (MCR) demands enhancements in the way change logs are presented to reviewers. Their task benefits from higher-level descriptions about the intention behind commits; for instance, evolution tasks, such as *refactorings*, could be more effectively reviewed in the presence of the intended transformation – better if automatically detected by MCR tool support. This paper presents a *systematic literature mapping* (combining results from search strings, snowballing and a two-phase classification scheme) on refactoring-aware solutions to support MCR. We could observe that, since 2015, interest in tools and techniques for automatic detection of refactorings has been steadily growing. Most publications refer to new development methods or carry out characterisation studies. As a consequence of this overview, we point out a few potential research topics for the next years. In special, detection of multiple refactoring types in a mixed change log (in which refactorings are combined with other kinds of changes), or the need for case studies or experiments in applying refactoring detection in MCR, on distinct application domains and development environments.**

***Index Terms*—refactoring-aware, code review, software evolution**

I. Introduction

Software development proceeds as a series of code edits [1], in particular in the agile scenario, in which requirements changes/evolution are numerous, and several rounds of code edits are performed during each development cycle [2]. In this context, code review is a well-established strategy for detecting faults and improving software quality. Modern Code Review (MCR) is a lightweight, and tool-based form for code reviewing that has been widely adopted in both industrial and Open Source Software (OSS) projects [3], [4]. Developers often apply code review aiming at assessing the quality of their code changes (patches) before committing them to a project's repository. Hence, MCR effectiveness depends on different factors, such as reviewers experience and knowledge regarding the system code, review tools, and a proper understanding of each code edit [5].

In practice, code reviewers and submitters join forces to find possible faults, discuss the code edits, and include comments. Those tasks are often assisted by a code review tool (e.g., *Gerrit*[1]). During MCR, reviewers need to explore several code elements trying to find possible ineffective/useless code; and revisit the *changeset* (set of code edits).

[1]https://www.gerritcodereview.com/

Refactorings – code transformations that should improve a software design and quality but preserve its external behaviour [6] – play an essential role since a significant part of the changeset are refactoring edits [7], [8]. Given that the quality of the change description, and nature of the change, significantly affect the code review effectiveness [9], research evidence suggests that a refactoring-aware review (in which, by whatever method, the reviewer is informed about the refactorings applied to a changeset) can help a reviewer improve his/her edit understanding, reduce reviewing time, and increase its accuracy [10]–[12]. In this sense, recent work on refactoring-aware solutions for MCR has proposed tools for automatic detection of refactorings edits in changesets [10], [13], [14].

Once *systematic literature mappings* provide a structured manner to obtain an overview of a research area, we carried out a study to gather evidence on the extent of the work (in terms of the existing support, research trends, and possible open research topics) related to refactoring-awareness and MCR. In this context, our study applies a systematic approach [15] over research questions focusing on topics, strategies, and contributions in the area. The main findings emphasise the following issues:

- Lack of appropriate characterisation studies, with a need for the application of the proposed techniques and tools in MCR contexts;
- Lack of accurate support to review patches with multiple refactorings (of different types);
- Need for further empirical studies on the effectiveness of the refactoring-aware solution for MCR, in both open source and industrial scenarios.

II. Research Method

Our systematic mapping aims at collecting research publications related to refactoring awareness and MCR. We understand refactoring awareness as any automatic approach/tool for distinguishing which changes are refactorings edits in a changeset, either classifying refactoring types or not.

For guiding our investigation, to obtain an overview of the state-of-the-art, trends and gaps, we define the following research questions:

- **RQ1**. What are the most common research topics?

978-1-7281-2270-0/19/$31.00 ©2019 IEEE
DOI 10.1109/IWoR.2019.00019

- **RQ2**. What are the methods/techniques/tools proposed?
- **RQ3**. What are the validation methods applied?

These RQs intend to figure out research questions addressed (RQ1), the types of research contribution, such as techniques and tools (RQ2), and the domain (OSS or industrial projects) and the research validation type. For instance, examples and empirical studies based on analysis, evaluation, or experience (RQ3).

We first applied a *search string* on well-established scientific databases, which were chosen driven by recommendations for Software Engineering, as suggested by Kitchenham and Brereton [16], including the ACM Digital Library, IEEEXplore Digital Library, Inspec & Compendex Guide, Science Direct, and Springer Link. Based on the research questions, and to work with a manageable number of studies, we defined the following search string:

("code review" OR "code inspection") AND ("refactoring detection" OR "refactoring" OR "tool" OR "approach" OR "method" OR "process" OR "technique")

For managing references and exclude repetitions, we used the Zotero tool[2]. This search resulted in more than 3,700 articles (including repetitions).

For working with a significant set of studies, we rigorously analysed the papers' title and abstract, to figure out only specific works related to refactoring-awareness and MCR. We also defined and applied the selection criteria listed in Table I. Our goal was to obtain an overview of the state-of-the-art, trends and potential research gaps of the area. Therefore, it is imperative that the selected studies should be available online and include empirical results.

TABLE I
INCLUSION AND EXCLUSION CRITERIA

Criteria	Description
Inclusion	Peer-reviewed papers describing empirical works. Studies published in Journals or Conferences, Workshops and Symposium proceedings. Written in English.
Exclusion	Lack of empirical results. Only available in the form of abstract or summary. Similarity to already selected research (the most recent paper was considered).

After this step, we collected seven (7) conference proceedings papers and three (3) journal articles. At this point, these ten manuscripts were used as a starting set for a backward snowballing procedure [17], that led to three (3) new conference papers [18]–[20]. This result suggests that the primary papers have been already collected by our search string, due to the high number of repetitions raised at snowballing. In particular, we have identified documents mostly related to initial work on area [18], [19].

To figure out the nature and contributions of the selected studies, we carefully read the abstracts focusing on their research questions, results, and validation methods. In the case of poorly-assessed abstracts, we also examined the introduction and conclusion sections. Hence, to categorise the thirteen selected papers, we employed a two-phase classification scheme. In the first phase, we were supported by Shaw's work [21], which specifies classifications for research questions, results (research contributions), and validation in Software Engineering (Table II).

TABLE II
RESEARCH QUESTIONS, RESULTS, AND VALIDATION CATEGORIES IN SOFTWARE ENGINEERING

Categories	Classification
Question	Method or means of development Method for analysis or evaluation Design, evaluation, or analysis of a particular instance Generalization and characterization Feasibility study or exploration
Results	Procedure or technique Qualitative or descriptive model Empirical model Analytic model Tool or notation Specific solution, prototype, answer or judgment Report
Validation	Analysis Evaluation Experience Example Persuasion Blatant assertion

A summary of Shaw's classification approach [21].

In the second phase, a classification of the papers was performed through a specific keywording process applied for research questions and results, obtained from the first phase. At keywording, a fine-grained classification was fulfilled in our scheme, as showed at the **Summary** column in Table III.

This study has a few threats to validity. Since most steps of the selection, extraction and classification processes were performed by the first author; it may generate a significant threat to our findings. However, to reduce this limitation, we ran snowballing followed by a two-phase classification scheme, even though relevant studies may still have been missed. Despite the impossibility of eliminating researcher bias, we involved two authors in the classification. We depended, of course, on well-written abstract, introduction and conclusion sections. Thus, although levels of classification were applied in the first phase of our classification scheme, it was challenging to achieve a correct classification, in cases of unclear manuscripts. Even considering the guidelines and supporting techniques, the researcher bias can also be present in the raised discussions and conclusions.

All steps of our systematic mapping process were described in this paper to enable possible replications. Full data and details can be found in our website[3].

[2]https://www.zotero.org

[3]https://github.com/flaviacoelho/racr-sysmap

III. Results and Discussion

Based on Table III, our results and related discussions are presented considering each of our research questions in line with the respective classification for the research question, results, and validation from Table II (**Type of Research** field).

1) What are the most common research topics?: From the middle of this decade, we can see a variable number of publications by year, with a higher concentration in 2015. The main topics under investigation are approaches for *decomposing changes, refactoring code review, and inspection of systematic changes*.

A changeset is composed of multiple change edits, and composite changes (also referred to as tangled changes [22]) encompass code changes that address diverse development issues in a single patch. Refactorings and bug-fixes are examples of composite changes. For instance, a *Move Class* refactoring is composed of two atomic change operations: excluding a class from a package and adding it to another package [18], [20]. Systematic changes, as characterised by Kim and Notkin [19], specifically comprise change edits to code elements that own properties in common, for instance accessing the same field. In this concept, refactorings are usually systematic (e.g., *Move Class*).

Concerning topics published from 2007 to 2018, we observe that research on *methods of development* has been predominantly performed since 2015, differently from the focus of initial research on *characterization* studies. Refactoring-aware for MCR has evolved from the need for understanding development sessions to monitor software evolution [18] and systematic code changes [19]. Then, from 2012 to 2014, empirical studies focused on the relationship between refactoring and software faults [23], the demand for refactoring-aware code review tools [24], and the need for understanding changes during MCR, mainly in case of multiple changes (e.g., refactorings and bug-fixes) in a single commit [8].

A change in focus was observed from 2015. First, the number of publications has increased, based on previous characterisation studies [8], [23], [24] emphasising the importance and challenges in MCR. From 2015, we found eight studies proposing new methods of development. This number suggests that as the research area advances, the number of development methods proposals also grows, compared to the characterisation studies. This scenario suggests a growing trend in research conducted for improving the quality and effectiveness of MCR.

In particular, our study found recent solutions in terms of systematic change inspection tools [25], composite change inspection tools [7], [26], and refactoring-aware code review and clone refactoring inspection tools [10], [13], [14]. Regarding systematic change inspection, Zhang and colleagues [25] evolve the proposal from Kim and Notkin [19] for inspecting systematic code changes and detecting potential anomalies. Their approach is based on templates, evolving the solution for interactive use.

2) What are the methods/techniques/tools proposed?: Since 2015, development methods have been mostly implemented as tools for refactoring-aware code review, and composite/systematic change inspection. At least one heuristic-based technique has been proposed for identifying composite changes [20].

Concerning refactoring detection, while some published results adopt a high-level view, in which refactorings are detected but not classified, on the other hand, some employ a low-level approach, in which refactoring types are summarised for supporting MCR. Ge and colleagues [10] propose an approach which highlights five types of refactoring. Alves and colleagues have increased the number of refactoring types, and go a step further by detecting incomplete refactorings, and providing a safe-driven inspection of possible extra edits along refactorings [14]. In this sense, Chen and colleagues [13] propose a specific pattern-based tool for clone refactoring inspection and detection of anomalies.

Decomposing complex changes for MCR were studied by Barnett and colleagues, who developed a technique for summarising regions of changes [7], while Guo and Song developed an approach to expand the composite changes decomposition for interactive use [26]. Also, Tao and Kim have presented a heuristic-based technique for decomposition into semantically cohesive change-slices [20].

3) What are the validation methods applied?: Most of the selected studies propose methods of development, while some report on characterisation studies. In the face of the nature of these contribution types, since 2015, we found that Experience is the most common validation method.

Hence, the impact on MCR effectiveness has been addressed as the main focus of the proposed techniques validation, through industrial case studies [8], [24], [25], as well as empirical experiments [10], [13], [19], [20], [26], and controlled experiments with practitioners [7], [14]. However, distinct and unmatched metrics have been applied. For measuring effectiveness, Tao and Kim [20] consider the time spend in reviewing tasks, while Chen and colleagues [13] apply usefulness. Accuracy is measured in terms of identifying related atomic changesets by Guo and Song [26], while Ge and colleagues [10] regard precision and recall. Besides, only Kim and Notkin [19], and Zhang and colleagues [25] perform comparative studies between their approaches and other similar techniques. This scenario suggests individual efforts to establish the effects of the new techniques for the area; thus, denoting the need for future research towards a pattern of measuring refactoring-aware code review effectiveness.

IV. Conclusion

Our findings suggest a few potential research directions. On refactoring-aware code review tools, we observed solutions are needed in addressing accurately multiple refactorings (of different types), as combined in a changeset. Our study reinforces the need for further empirical studies and experiences, such as user studies with practitioners (submitters and reviewers), on the effectiveness of code review process driven by the current refactoring-aware tools, on different application domains at OSS, and industrial environments. Replication of studies is

TABLE III
SYSTEMATIC MAPPING RESULTS

Selected Paper	Venue	Year	Summary	Type of Research		
				Question	Result	Validation
Robbes and Lanza [18]	ICPC (C)	2007	Development sessions understanding	Ch	D	Ev
Kim and Notkin [19]	ICSE (C)	2009	Systematic change inspection	M	T	Exp
Bavota and colleagues [23]	SCAM (C)	2012	Refactoring and software defects	Ch	A	An
Tao and colleagues [8]	FSE (C)	2012	Change understanding	Ch	A	Ev
Kim and colleagues [24]	IEEE TRANS SOFTW ENG (J)	2014	Refactoring challenges and benefits	Ch	A	Ev
Zhang and colleagues [25]	ICSE (C)	2015	Systematic change inspection	M	T	Exp
Matsuda and colleagues [22]	IWPSE (C)	2015	Refactoring-aware change reorganizing	M	T	Exa
Tao and Kim [20]	MSR (C)	2015	Decomposition of composite change	M	Te	Exp
Barnett and colleagues [7]	ICSE (C)	2015	Decomposition of composite change	M	T	Exp
Ge and colleagues [10]	VL/HCC (C)	2017	Refactoring-aware code review	M	T	Exp
Guo and Song [26]	COMPSAC (C)	2017	Decomposition of composite change	M	T	Exp
Alves and colleagues [14]	IEEE TRANS SOFTW ENG (J)	2018	Refactoring-aware code review	M	T	Exp
Chen and colleagues [13]	J SOFTW-EVOL PROC (J)	2018	Clone refactoring inspection	M	T	Exp

Venue: C (Conference proceedings), J (Journal article)
Question: Ch (Characterization), M (Method of development)
Result: A (Answer), D (Descriptive model), Te (Technique), T (Tool)
Validation: An (Analysis), Ev (Evaluation), Exa (Example), Exp (Experience)

also relevant. Topics to be investigated should include the size and complexity of code changes, automation issues, diversity in refactoring goals, and code reviewer experience.

REFERENCES

[1] A. Mockus, D.M. Weiss and M. Song, "Predicting risk of software changes," Bell Labs Technical Journal, vol. 5, issue 2, 2000, pp. 169–80.

[2] J. Shore, "The Art of Agile Development: Pragmatic guide to agile software development," O'Reilly Media, Inc., 2007.

[3] A. Bacchelli and C. Bird, "Expectations, Outcomes, and Challenges of Modern Code Review", in International Conference on Software Engineering (ICSE'13), USA, 2013, pp. 712–21.

[4] T. Baum, O. Liskin, K. Niklas and K. Schneider, "Factors Influencing Code Review Processes in Industry," in 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'2016), USA, 2016, pp. 85–96.

[5] S. Mcintosh, Y. Kamei, B. Adams and A.E. Hassan, "An Empirical Study of the Impact of Modern Code Review Practices on Software Quality," Empir. Softw. Eng., vol. 21, ed. 5, 2016, pp. 2146–89.

[6] M. Fowler, "Refactoring: Improving the Design of Existing Programs," Addison-Wesley, 1999.

[7] M. Barnett, C. Bird, J. Brunet and S. K. Lahiri, "Helping Developers Help Themselves: Automatic Decomposition of Code Review Changesets, " in 37th International Conference on Software Engineering (ICSE'15), USA, 2015, pp. 134–44.

[8] Y. Tao, Y. Dang, T. Xie, D. Zhang and S. Kim, "How Do Software Engineers Understand Code Changes? An Exploratory Study in Industry," in ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE'12), USA, 2012, pp. 1–11.

[9] A. Ram, A.A. Sawant, M. Castelluccio and A. Bacchelli, "What Makes a Code Change Easier to Review: An Empirical Investigation on Code Change Reviewability," in 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'18), USA, 2018, pp. 201–12.

[10] X. Ge, S. Sarkar, J. Witschey and E. Murphy-Hill, "Refactoring-aware code review," in IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'17), USA, 2017, pp. 71–9.

[11] N. Tsantalis, M. Mansouri, L.M. Eshkevari, D. Mazinanian and D. Dig, "Accurate and Efficient Refactoring Detection in Commit History," in 40th International Conference on Software Engineering (ICSE'18), USA, 2018, pp. 483–94.

[12] T. Baum, and K. Schneider, "On the Need for a New Generation of Code Review Tools," in Product-Focused Software Process Improvement (PROFES), organized by P. Abrahamsson, A. Jedlitschka, A.N. Duc, M. Felderer, S. Amasaki, and T. Mikkonen, Lecture Notes in Computer Science, Springer International Publishing, 2016, pp. 301–8.

[13] Z. Chen, Y. Kwon and M. Song, "Clone refactoring inspection by summarising clone refactorings and detecting inconsistent changes during software evolution," J. Softw.-Evol. Proc., vol. 30, issue 10, 2018, pp.e1951–n/a.

[14] E. L. G. Alves, M. Song, T. Massoni, P.D.L. Machado, and M. Kim, "Refactoring Inspection Support for Manual Refactoring Edits," IEEE Trans. Softw. Eng., vol. 44, issue 4, 2018, pp. 365–83.

[15] K. Petersen, S. Vakkalanka and L. Kuzniarz, "Guidelines for Conducting Systematic Mapping Studies in Software Engineering: An Update," Inf. Softw. Technol. USA, vol. 64, issue C, 2015, pp. 1–18.

[16] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," Inf. Softw. Technol. USA, vol. 55, issue 12, 2013, pp. 2049–75.

[17] C. Wohlin, "Guidelines for snowballing in systematic literature studies and replication in software engineering," in 18th International Conference on Evaluation and Assessment in Software Engineering (EASE'14), USA, 2014, pp. 38:1–10.

[18] R. Robbes and M. Lanza, "Characterizing and Understanding Development Sessions," in 15th IEEE International Conference on Program Comprehension (ICPC'07), Canada, 2007, pp. 155–66.

[19] M. Kim and D. Notkin, "Discovering and Representing Systematic Code Changes," in 31st International Conference on Software Engineering (ICSE'09), USA, 2009, pp. 309–19.

[20] Y. Tao and S. Kim, "Partitioning Composite Code Changes to Facilitate Code Review," in 12th Working Conference on Mining Software Repositories (MSR'15), USA, 2015, pp. 180–90.

[21] M. Shaw, "Writing good software engineering research papers: mini tutorial," in 25th International Conference on Software Engineering (ICSE'03), USA, 2003, pp. 726–36.

[22] J. Matsuda, S. Hayashi and M. Saeki, "Hierarchical Categorization of Edit Operations for Separately Committing Large Refactoring Results," in 14th International Workshop on Principles of Software Evolution (IWPSE'15), USA, 2015, pp. 19–27.

[23] G. Bavota, B. De Carluccio, A. De Lucia, M. Di Penta, R. Oliveto and O. Strollo, "When Does a Refactoring Induce Bugs? An Empirical Study," in IEEE 12th International Working Conference on Source Code Analysis and Manipulation (SCAM'12), Italy, 2012, pp. 104–13.

[24] M. Kim, T. Zimmermann and N. Nagappan, "An Empirical Study of Refactoring Challenges and Benefits at Microsoft," IEEE Trans. Softw. Eng., vol. 40, issue 7, 2014, pp. 633–49.

[25] T. Zhang, M. Song, J. Pinedo and M. Kim, "Interactive Code Review for Systematic Changes," in 37th International Conference on Software Engineering - Volume 1 (ICSE'15), USA, 2015, pp. 111–22.

[26] B. Guo and M. Song, "Interactively Decomposing Composite Changes to Support Code Review and Regression Testing," in IEEE 41st Annual Computer Software and Applications Conference (COMPSAC'17), Italy, 2017, pp. 118–27.