An Empirical Study on Research and Developmental Opportunities in Refactoring Practices

Shivani Jain
University School of Information, Communication and Technology, GGS Indraprastha University
Sector 16 C, Dwarka
Delhi, India
shivani.1091@gmail.com

Anju Saha
University School of Information, Communication and Technology, GGS Indraprastha University
Sector 16 C, Dwarka
Delhi, India
anju_kochhar@yahoo.com

***Abstract*—Maintaining large complex software is one of the major challenges faced by today's software industry. Refactoring is one way to do so. It is the process of changing internal structure of project code or software design without altering functionality. It improves software quality and reduces software entropy. This paper presents the preliminary results of an explanatory survey targeted at investigating refactoring practices by IT professionals. 221 participants helped to reveal important facts about refactoring risks, benefits, limitations of tools, and how a team manages consistency between different artefacts while practising refactoring. Findings reveal that refactoring tools are under-used as they have availability, usability and trust issues. An automated system is the need of the hour to manage change consistencies, visualizing code structures, detecting code, and design smells, and performing refactorings. This study will enable researchers and developers to understand their role in a better way as prevailing issues with current state-of-art are exposed and challenges are reported.**

Keywords: Refactoring; code smells; refactoring tools; survey; empirical study.

I. INTRODUCTION

Code smells are design flaws, though are free of syntax errors, but can lead to future bugs and errors [1]. They are a violation of basic design principles and are also known as anti-patterns [2]. They contribute to financial debts and make software complex, hard to understand and maintain, and make changes difficult to embed. Thus, detection of such designs flaws and their correction is an absolute necessity [3]. Refactoring is one of the techniques to remove these anomalies. Refactoring is a well-organized practice for the reorganization of the present body of code, changing its internal structure without changing its external conduct. It is a sequence of little functionality preserving transformations. It is done while adding a feature, fixing a bug, and during code review [4].

Refactoring is mainly done to improve the internal structure and readability of the software. It increases flexibility, maintainability and reduces inter-modular couplings [5, 18]. It can be done by a specialist or stakeholders like software designer, developer, tester or maintenance team. It can be done either manually or with the help of a tool. The scope can be system-wide or small scale, depending on the aim of applying refactorings. Refactoring is a time-consuming process that does not reflect immediate benefits like new features or bug fixes [6]. Incomplete and incorrect refactorings can lead to bugs [7] and it has been found that a high proportion of refactorings often led to an increase in the number of errors [8, 17].

The refactoring process [9] consists of following activities: (1) identify code smells; (2) determine which refactorings are best suitable for application; (3) make sure that applied refactoring preserves behaviour; (4) apply refactorings; (5) calculate the impact of refactorings on software quality features; (6) maintain uniformity between refactored program code and other software relics (test data, documents etc.).

Murphy-Hill [10] mentioned four different ways to collect research data for refactoring. They are:

- Mining the Commit Log – Look for mention of the word "refactor" in the commit logs of versioned repositories. Commit logs are updated when a programmer commits a change to the repository.
- Analyse Code Histories - Analyze an order of versions of the source code by manual comparison or by automating the comparison using a software tool.
- Observing Programmers - Observe developers in the field, working on software development and illustrate their refactoring behaviour. Such observation can be direct observation which comes under the category of a controlled experiment. Another is indirect observation which can be a survey or a project post-mortem.
- Logging Refactoring Tools - Some programming environments automatically record the programmer's activity in a log file. Such an environment is specialized to collect refactoring tool events.

DOI reference number: 10.18293/SEKE2019-038

In this study, the third method i.e. indirect observation has been implemented and data has been collected through a survey.

In this study, the following research questions have been addressed:

RQ1: How do programmers ensure that code has been refactored correctly and what are the measures taken to manage consistency between software artefacts?

RQ2: What are the reasons that prompt the refactoring process?

RQ3: What are the common refactoring practices followed by developers?

RQ4: Which are the most desirable features and barriers in the adoption of refactoring tools?

RQ5: What are the risks and benefits associated with refactoring?

To find answers to these burning questions, a survey was conducted and 221 software engineers participated in the study. Responses were collected online and analyzed quantitatively. The results were presented pictorially through graphs.

This study will make the following contributions:

- Study will assist researchers to identify research areas in the field of refactoring and to focus on issues to be solved in refactoring process. It will support the developers to develop the tools keeping in mind the shortcoming of the available tools. And will help IT professionals to understand the importance of refactoring and make it a part of their development process in various projects.
- Learning limitations of refactoring tools will guide researchers to focus on grey areas and what are prominent research areas, for example validation and verification of applied refactorings, maintaining change consistency in between artefacts, and development of better algorithms for detection of code smells etc. Developers can build easy to understand tools with better GUI and work on availability issues as well by creating awareness among development community.

II. RELATED WORK

G. H. Pinto and F. Kamei [11] did a qualitative and quantitative study to find out answers to following research questions: Which are the most desirable features in refactoring tools? What are the factors that prevent developers to adopt refactoring tools? Does interest in refactoring tools increase over the years? To uncover the number of issues regarding these tools, more than 1,400 messages – 324 questions and 1,115 answers to those questions were analyzed from more than 1,200 users. Major findings of this study are: refactorings tools are in demand for dynamic languages, databases and multi-language refactorings. Users reported that a lack of trust and usability problems in tools still prevails. Interest in refactoring tools over the years has not increased as expected.

M. Kim, T. Zimmermann and N. Nagappan [12] conducted a survey at Microsoft, followed by a semi-structured interview and quantitative analysis of version history data of Windows 7 to reveal refactoring benefits and challenges. Survey finds that the refactoring definition in practice is not restricted to a standard definition of behaviour-preserving code alterations and developers observed that refactoring involves considerable cost and risks. The quantitative analysis of Windows 7 version history finds refactoring top 5% of modules led to a reduction in modular couplings and many complexity measures but increases the size more than the bottom 95%.

N. Singh and P. Singh [16] performed a comprehensive sentiments analysis on 3,171 GitHub comments during refactoring 60 open Java source projects by mining relevant commit messages. Research Question – "Does a refactoring task allocated during the implementation of a software feature following a strict deadline invoke positive or negative sentiments in the developer?" was answered. Tool RefTypeExtractor for automatically linking commit messages to their respective refactoring techniques was developed and dataset SentiRef, which stores the identified developer's sentiments linked to each of 3,171 commit messages was created. The research concluded that in general software developers express more negative sentiments than positive sentiments while performing refactoring tasks which reveals the substandard state of the refactoring process.

Arcoverde, Roberta, Alessandro Garcia, and Eduardo Figueiredo [13] presented the results of a survey with the purpose of understanding the longevity of code smells in software projects. They concluded that (i) there is a probability of breaking APIs before refactorings by developers of widely-scoped reusable code; (ii) developers of standalone applications consider contract breaking changes easier to apply than developers of reusable assets; (iii) refactoring tools are more frequently used by developers that apply Test-Driven Development; (iv) refactoring tools are commonly used, and (v) reusable assets and standalone applications have different refactoring prioritization.

Our study intends to foster such previous investigations by revealing current challenges faced by developers and how they maintain consistency between different artefacts. We designed a questionnaire in order to understand the gap that subsists between the interpretation of refactoring practices by developers and researchers. The questionnaire was made available as an online survey and 221 software engineers filled it.

III. SURVEY SETTINGS

In order to understand the refactoring practices, a questionnaire was created and sent to 10 experts (well learned software engineers in the IT industry with more than 12 years

of experience in companies like Amazon, Adobe, Flipkart, TCS, Walmart etc.) and based on their feedback, the questionnaire was refined regarding the clarity and objectivity of the questions. It consisted of 14 multiple choice questions and 6 free-form questions. Few multiple choice questions had an option where participants could write answers of their choice as well. A glossary was included at the beginning explaining terms and acronyms used, for disambiguation. To collect information online survey was conducted and 221 engineers participated. Participants belonged to different companies and various designations. Majorly, they were developers (i.e. 90.5%) including requirement engineers, software designers, testers, researchers, full stack developers, and software architects etc. having a maximum of 20 years of experience, minimum of a year and an average of 3.2 years. The survey was divided into four sections and is described in Table 1.

The questions were formulated to identify how often and when refactorings are performed, what is the main purpose behind it and how consistency is managed between different artefacts while performing refactorings. Another section was focused on the most popular tools and what are the desirable features and barriers in the adoption of refactoring tools. Further, investigation on benefits, risks, and challenges regarding refactoring is explored. After the collection of responses, data was analyzed and categorized pictorially. Results that were revealed were both interesting and useful in understanding the roles and responsibility of researchers and developers in the field of refactoring. It revealed major challenges that still prevail and scope in the research area.

IV. RESULTS

The survey was made online and 221 IT professionals participated. 90.5% of them were developers from different companies and having experience in various languages like Java, Python, .NET etc. 23.4% of participants performed refactoring daily, 34.9% weekly and 21.1% daily. Software engineers from diverse and virtuous companies like Amazon, HCL, Infosys, Flipkart, Oracle, TCS, Expedia, Snapdeal, IBM, Paytm, ISRO, and ICAR etc. contributed to our findings. Participants had an average experience of 3.2 years with a maximum of 20 years and working in various nations like India, USA, Australia, Germany, and China.

The following section organizes the results in terms of the research questions.

RQ1: How do programmers ensure that code has been refactored correctly and what are the measures taken to manage consistency between software artefacts?

Results show that 78% of participants perform some kind of testing after refactoring code. Most of them prefer simple unit or functional testing but some of them prefer regression, integration, smoke, sanity or boundary value testing. Few of them make their peers to do code review. To maintain consistency, teams use version control platforms like Git, communicate to the team through a pre-defined channel or by adding comments, maintain an excel file of changes. Test code and documents are changed manually after refactorings. Some of the remarks are as follows:

- "*Version control helps to keep a track of changes made, which once are completed successfully are documented.*"
- "*Rewrite/Update Unit Test, Update Documentation, Add relevant comments*"
- "*Code and test cases must go hand in hand. To ensure this I follow test based approach with unit test cases written before the code has to be refactored.*"
- "*We use XML notation for commenting code which reduces the need for separate documentation to a very large extent, code coverage tools for maintaining test cases.*"

Some of the good practices followed by engineers after performing refactorings are doing code reviews, compiling the code, running test cases, running bug detectors and modifying test cases according to the refactored code.

RQ2: What are the reasons that prompt the refactoring process?

Refactoring software is only beneficial when it is done with a purpose like reducing coupling between modules. Results revealed some of the reasons that prompt software engineers. 79% and 83% of professionals refactor when code gets hard to understand and maintain respectively. 50% agreed that slow performance and wide dependencies between modules are the main reason behind their refactoring actions. Logical mismatch, difficulty in debugging and testing, readability, re-usability and duplicity were also the main causes to initiate refactoring as depicted in Fig 1.

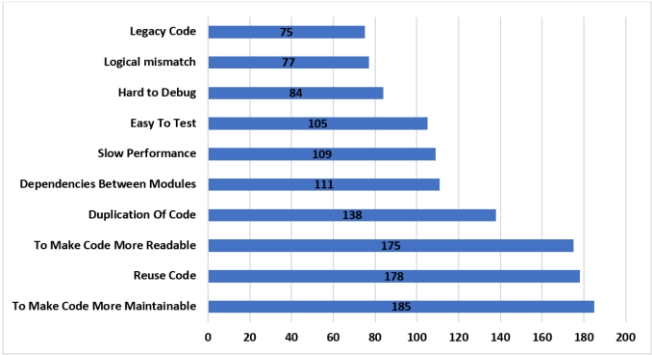

Figure 1. Reasons that prompt team to refactor code

Majority of the participants agreed that refactoring increases program flexibility, improves readability, reduces coupling, improves the internal structure of the code and makes it easier to add new features. Apart from these, honourable mentions were to reduce bugs, to increase consistency of an application, to reduce compiling time, to optimize and improve software performance, to enhance scalability and robustness of applications.

TABLE I. Summary of Survey Questions

Background	Which best describes your primary work area (developer, tester, manager etc.)? (open answer)
	How many years of work experience do you have in the software industry? (open answer)
	Name of Current Company and Country (open answer)
Refactoring Practices	How often do you perform refactoring (Daily, Weekly, Monthly, Yearly, Seldom, Never)? (multiple choice)
	Which keywords do you use or have you seen being used to mark refactoring activities in change commit messages? (multiple choice)
	How do you ensure that you have refactored program correctly? (open answer)
	How do you manage consistency between different software artefacts (e.g. documents, code and test cases) during refactoring? (open answer)
	What is the purpose of your refactorings? (multiple choice with the open answer)
	Which of these reasons prompts you to initiate the refactoring process? (multiple choice with the open answer)
	Select following options for refactorings [multiple choice: (a) manually and with a tool (b) manually, (c) using automated tools, (d) know this but don't use it, (e) don't know this refactoring.] • Rename, Extract Method, Encapsulate Field, Extract Interface, Remove Parameters, ... [From Fowler's catalogue]
	How do you strongly agree, agree, neither agree or disagree, disagree, strongly disagree with each of the following statements? • I perform refactorings with other functional changes. • Refactorings I want to perform are different from what supported by tools. • Tools do not support higher level refactorings. • How do you validate code refactorings? Few statements are shown in this table for presentation purposes.
Refactoring Tools	What tools do you use during refactoring? (open answer)
	How do you perform most of your refactorings? (multiple choice with the open answer)
	What are the barriers to adoption of refactoring tools? (multiple choice with the open answer)
	What are the features in refactoring tools you would like to have? (multiple choice with the open answer)
Risks and Benefits	How do you strongly agree, agree, neither agree or disagree, disagree, strongly disagree with each of the following statements? • Refactorings advance code readability • Refactorings introduce subtle errors • Refactorings disrupt other programmer's code • Refactorings advance performance • Refactorings make it debugging easy.
	What are the challenges associated with performing refactorings? (open answer)
	Based on your own experience, what are the risks involved in refactoring? (multiple choice with the open answer)
	What benefits have you observed from refactoring? (multiple choice with the open answer)
Only some of the questions are mentioned for representation purpose.	

RQ3: What are the common refactoring practices followed by developers?

Great proportion strongly agreed on the following practices:

- Refactorings are carried out in batches and changes in associated test cases and documents are reflected.
- Refactorings are done with other types of changes which modifies program behaviour externally. Pure refactorings are hardly done. This observation is consistent with R. Johnson's study [14].
- Refactorings that are done manually differ from what tools offer.
- Refactorings that are applied are higher level changes which are not supported by tools. This informs about the need for tools for higher level refactorings, for example dealing with generalization refactorings.
- Majority of the refactorings (60.6%) are done manually. This practice proves the urgent need for good quality, available and easy to use tools.
- Renaming, Extract Method and Remove Parameters are the most common refactorings performed manually or with the help of a tool. The same observation was made by M. Kim [15].

RQ4: Which are the most desirable features and barriers in the adoption of refactoring tools?

Participants listed a wide variety of tools that are used to perform common refactorings. Most commonly used tools are Jenkins, ReSharper, CodeRush, JS Refactor, Visual Assist X, TSLint, DPack, JetBrain etc. Refactorings like renaming and move method are simply done in IDEs like Eclipse, IntelliJ, Visual Studio etc.

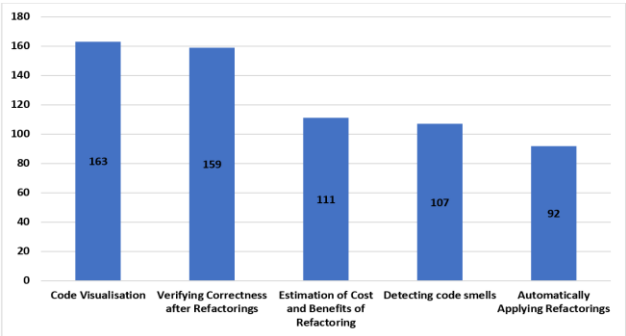

Figure 2. Most desirable features of refactoring tools

Most desirable features, developers want in refactoring tools are code smell detection feature, 74% want code visualization applicability. 72% of professionals would like to verify correctness feature after they are done performing refactorings, estimation of cost and benefits of refactoring are another requirement that participants mentioned. Automatically applying refactorings was only suggested by 42%. Fig 2 represents data in graphical form.

65% stated barriers that cease developers in adopting tools are less or no knowledge about the availability of tools. Around 29% participants mentioned difficult to understand/learn tools and unknown or not able to understand the debugging process is their reason that prevents them to use refactoring tools. Tools are not trustworthy and have bad GUI. Fig 3 represents same. Other mentions were:

- *"Company support"*
- *"Languages like Python and CSS have limited support for refactoring tools."*
- *"They sometimes don't understand that "why I do, what I do" such as if I declare something in multiple lines, I mean it to be so, but formatting online size makes it hard to understand."*

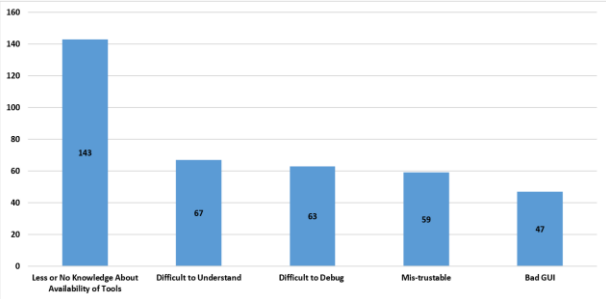

Figure 3. Barriers in the adoption of refactoring tools

RQ5: What are the risks and benefits associated with refactoring?

Refactoring risks are quantitatively presented through the graph in Fig 4. Key causes are identifying code smells. It is one of the major research areas in the field of refactoring. Many automatic, semi-automatic, and metric-based code smell detection techniques have and are being developed. Managing consistency between artefacts is a long and regressive process. Managing time is challenging as developers avoid refactoring code before major releases. Refactorings might introduce bugs or break existing code. Preserving behaviour, understanding legacy code, and convincing management team are considerable challenges.

Refactoring though is time-consuming procedure but it definitely yields promising benefits such as improved maintainability and readability which was further supported by more than 80% of the participants in this empirical study. More than 60% acknowledged improved performance, reduction in code size as well as duplicate code are perks of refactoring code. Other advantages are the reduction in release time and bugs, software becomes easy to test and add the new features. Fig 5 represents the benefits of refactoring pictorially through a bar graph.

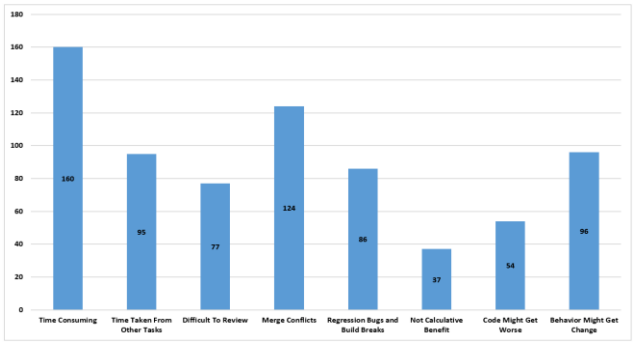

Figure 4. Risks involved in the refactoring process

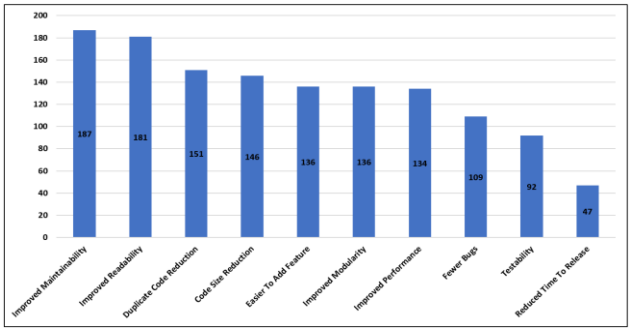

Figure 5. Benefits of performing refactorings

V. THREATS TO VALIDITY

Some points to be considered are:

Participants of survey conducted were IT professionals not refactoring specialists and with the assumption that people who filled the survey know what "Refactor" means. Survey made no inquiry about the type of projects (e.g. Web applications, Embedded systems, Information systems, etc.) participants had experience in. A number of participants were low to generalize the results and few of them had the experience below 5 years. Most of the questions were close-ended which might lead to biasing.

VI. CONCLUSIONS AND FUTURE WORK

Large scale survey was conducted and a wide range of IT professionals was engaged. The main purpose of the survey was to understand the trends followed by developers and what are the research opportunities in the field of refactorings and developmental challenges. The study also answers the questions like what sort of tools should be built to better or automate the whole process and limitations of currently available tools. Survey responses were collected and analyzed to conclude the following results:

The refactoring process needs automated tools to maintain changes between different artefacts of software.

Some companies use version control systems but a system that updates the design, test cases, documentation automatically after refactoring is in demand. Software companies can invest their efforts in this sector.

Refactoring without purpose will not yield any benefits. Some of the factors that prompt refactorings are - hard to understand and maintain program code, wide inter-modular dependencies, difficulty in debugging and testing, readability issues etc. Software development teams should devote their time in refactoring process to overcome such issues.

Refactoring increases program flexibility, reduces coupling, improves the internal structure of the code and makes it easier to add new features. These points were supported by the majority of participants. Benefits associated with the refactoring process are improved maintainability, performance, modularity and readability, reduction in code size, duplicate code, release time and bugs, and easy to test. They are the motivations for teams to invest more time and effort in the refactoring process.

Common practices in refactoring are that they are carried out in batches, done with other types of changes that modify external program behaviour. Majority of the refactorings (60.6%) are done manually. Renaming, Extract Method, and Remove Parameters are the most common refactoring performed manually or with the help of a tool.

The reason that prevents software engineers to adopt refactoring tools is - less or no knowledge about the availability of tools, difficult to understand/learn, unknown or not able to understand debugging process, trust issues, and bad GUI. So, developers and researchers should investigate the reasons behind such an inappropriate condition of refactoring tools and try diminishing them. Tools for many dynamic languages are still unavailable. Code structure visualization, code smell detection, cost and effort estimation tools are coveted. Tools to validate and verify refactorings need to be developed.

Most desirable features, developers want in refactoring tools are automatic code smell detection, code visualization aspect, verifying correctness after refactoring edits, estimation of cost, efforts and benefits of refactoring, automatically applying refactorings. This provides research areas that should be explored by researchers and developers working in the field of refactoring.

Refactoring challenges faced by professionals are identifying code smells, managing consistency between artefacts, time management, excessive couplings between modules, refactorings might introduce bugs or break existing code, behaviour preservation, understanding legacy code, and convincing management team. Researchers can work in these domains to ease up the refactoring process. Companies and teams should realize the importance of refactoring code and refactoring activities needs to be further encouraged to reduce maintenance time, money, and effort.

For future work, personal interview with professionals can be conducted on a large scale and further conclusions can be made by observing refactorings patterns and behaviour of IT teams in the field.

REFERENCES

[1] V. Maggio and M. Faella, “Improving the Design of Existing code,” Slides, 2011.

[2] T. Mens and T. Tourwé, “A survey of software refactoring,” IEEE Trans. Softw. Eng., vol. 30, no. 2, pp. 126–139, 2004.

[3] M. Tufano et al., “When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away),” IEEE Trans. Softw. Eng., vol. 43, no. 11, pp. 1063–1088, 2017.

[4] M. Fowler, “Refactoring,” p. 13472, 2002.

[5] W. Opdyke, “Refactoring Object-Oriented Frameworks,” PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[6] W. Opdyke, "Refactoring, reuse & reality," Lucent Technologies/Bell Labs, 1999.

[7] C. G€org and P. Weißgerber, “Error detection by refactoring reconstruction,” in Proc. Int. Workshop Mining Software Repositories, 2005, pp. 1–5.

[8] P. Weißgerber and S. Diehl, “Are refactorings less error-prone than other changes?,” Proc. ACM Int. Workshop Mining Software Repositories, 2006, pp. 112–118.

[9] A. V. D. Tom Mens, “Refactoring: Emerging Trends and Open Problems,” IEEE Int. Conf. Software Maintenance, ICSM, pp. 521–522, 2003.

[10] E. Murphy-Hill, Danny Dig, Chris Parnin, “Gathering refactoring data: a comparison of four methods,” Proc. 2nd Work. Refactoring Tools WRT 08 conjunction with Conf. Object Oriented Program. Syst. Lang. Appl. OOPSLA 2008, 2008.

[11] G. H. Pinto and F. Kamei, “What programmers say about refactoring tools?,” Proc. 2013 ACM Work. Work. refactoring tools - WRT ’13, pp. 33–36, 2013.

[12] M. Kim, T. Zimmermann, and N. Nagappan, “An Empirical Study of Refactoring Challenges and Benefits at Microsoft,” IEEE Trans. Softw. Eng., vol. 40, no. 7, pp. 633–649, 2014.

[13] Arcoverde, Roberta, Alessandro Garcia, and Eduardo Figueiredo. "Understanding the longevity of code smells preliminary results of an explanatory survey." Proceedings of the 4th Workshop on Refactoring Tools. ACM, 2011.

[14] R. Johnson, “Beyond behavior preservation,” Microsoft Faculty Summit 2011, Invited Talk, Jul. 2011.

[15] Kim, Miryung, Thomas Zimmermann, and Nachiappan Nagappan. "A field study of refactoring challenges and benefits," Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM, 2012.

[16] Singh, Navdeep, and Paramvir Singh, “How Do Code Refactoring Activities Impact Software Developers' Sentiments?-An Empirical Investigation Into GitHub Commits,” 24th Asia-Pacific Software Engineering Conference, pp. 648-653 IEEE, 2017.

[17] Alshayeb, Mohammad, "Empirical investigation of refactoring effect on software quality," Information and software technology 51.9 (2009), 1319-1326.

[18] Szőke, Gábor, Gábor Antal, Csaba Nagy, Rudolf Ferenc, and Tibor Gyimóthy, "Empirical study on refactoring large-scale industrial systems and its effects on maintainability," Journal of Systems and Software 129 (2017): 107-126.