DOI: http://dx.doi.org/10.26483/ijarcs.v9i5.6324

Volume 9, No. 5, September-October 2018

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

ANALYZING REFACTORING TRENDS AND PRACTICES IN THE SOFTWARE INDUSTRY

Zeba Khanam
Assistant Professor
Saudi Electronic University
Dammam, KSA

Abstract---Most of the developers in the software industry though have some instinctive capacity to refactor but the true expertise of this skill is rarely found in real time software development. Thus it is significant to discern the reasons as to how prevalent are the refactoring practices in reality are and what are the major factors impacting its role and adoption in software development. This paper explores the actual implementation of refactoring practices in software industry by taking inputs from the software professionals working in different projects and companies. This exploration is basically required to assess the gap between research practices and Industrial norms. The paper also tries to probe the consequences in software development in the absence of refactoring, what are the code rejuvenation practices adopted by the professionals when not refactoring or due to non familiarity or lack of expertise in the area.

Keywords-Refactoring, code smells, automated refactoring, manual refactorings, industry software practices, agile methodology.

I. INTRODUCTION

Comprehension exercises on the various existing software engineering (SE) practices and techniques used in industry are significant as it gives a real insight into the applicability of the theoretical theories and practices evolving rapidly. There exists a wide range of software practices and their types that are imbibed by the industry.

The drive for conducting the research on this topic is derived from various sources. First, refactoring has gained momentum in the industrial practices, along with widespread research and the rapid increase of tool support for automated refactoring [12][13] . A significant question in this context would be:
Can the benefits of refactoring be quantified and, if so, is it worth doing refactoring? What is the status of refactoring in the software industry? By exploring various works on refactoring, there has only been a confined research to highlight the role of refactoring in evolving commercial software and little research to establish the relevance of the refactored code with ease of maintenance or fault-proneness.

The other aspect that the paper tries to explore is the consequences resulting in the absence of refactoring. What are the code rejuvenation practices adopted by the professionals in absence of refactoring or due to non familiarity or lack of expertise in the area. How does it impact the software development process and creation of technical debt? These explorations would to an extent assist in bridging the gap between research practices and Industrial practices. Thus it intends to grasp and identify high level view on Software Engineering and maintenance Practices primarily used in the industry. A lot of research has already been conducted on refactoring techniques and tools. Most of the researchers have claimed that it helps in easing up the maintenance process. In another research it was found that not just the software metrics but the type of design the developer perceives is what influences the process of refactoring and at times the crosscutting concerns becomes a concern and need to be refactored [21][23]. One of the works suggested that the code smells and its severity may vary from one developer to the other and the significance of the class with the code smell also varies according to different developers as the code evolves [17][20]. Thus a trade-off between the quality and robustness is worth calculating before refactoring. Automated refactoring too have been explored extensively by the researchers and various approaches are already invented using search based techniques to assist in searching for a better design [24].Another important impact that has been explored by a few researchers lately is the improvement in the energy efficiency of the software due to refactoring [2][25][26]. The carbon footprint contributed by the ICT is around 2% of the total emission [25][27] . Refactoring has led to reduction in code smells and a better software design contributing to improved energy efficiency in ICT. Though this aspect has not been investigated in this paper but it would be explored in the future work.

This drove us to concentrate on two main objectives that are:

- Enquire and analyze the major refactoring trends in the industry
- What maintenance strategy is used to improve the software
- If not refactored then how the code does survive recursive modifications.

II. METHODOLOGY

The Delhi NCR region in India has a vibrant software industry and is worth exploring the Software engineering practices and in particular the status of refactoring. Our objective is to get a glance of the high-level view on type of

SE practices in an Industrial setup. To investigate the status of refactorings in the SE practices, we drafted an online and face to face survey with 10 questions based on the experience of the developer and his role.

The questionnaire that had been designed had the following queries:

1. Do you perform refactorings? On which platform and how often.
2. Manual or automatic? Which tool is preferred?
3. How big is the team and what is your role?
4. Which refactorings are most commonly done by your team?
5. Who usually performs the refactorings?
6. Any difficulty in assessing which is the refactoring to apply?
7. Which part of the code is generally most refactoring prone?
8. In the absence of refactoring what code upgradation technique is applied?
9. Which software development practice is the project following? Ever used Test Driven Development?
10. Any measurable result obtained by refactoring?

III. ANALYSIS AND RESULTS

This paper elaborates the significance and the status of refactoring in the software industry by performing the survey of refactoring practices and attitudes in different companies working on different domains in different teams. The study explores differences in expertise and attitudes about refactoring among participants who played roles in software development, and how these differences affected the actual practice. Many authors who had conducted research on the impact of refactorings claim substantial improvement in maintainability [1][9][10]. The study found that though refactoring at different stages and the roles is endorsed by the participants but full bloom refactoring with an in-depth knowledge about the code smells is very rare and refactoring needs to be practiced more often as there is a strong agreement about the negative impacts of deferring refactoring on the overall project.

We had carried out a large-scale empirical study that explored the views and experiences of approximately 80 experienced practitioners with regards to the prevalence of refactoring activities in mainstream development. We had conducted face to face interactions as well as email-based semi-structured interviews and had analysed different parameters impacting the adoption of this practice.

A. Absence of a Standard Strategy

The survey revealed that there are no strategies in place for implementation of refactoring techniques. No plans designed to guide the developers with the same. Developers performing with their own knowledge and sometimes ended up messing with the code. This indicates that the introduction to the code smells [15] should be made mandatory to the development teams. Though the agile team, practicing the test driven development had different perspectives and a deeper understanding of the phenomena [3][4]. Analysis of the survey results have raised many interesting questions suggesting the need for a considerable amount of future research.

B. Analysis of the Feedback

The developers are indulged in refactoring but somewhat in an informal state though the ones whom we surveyed had knowledge about the topic but not good enough to implement it practically. The survey was conducted on 5 teams (named as A, B, C, D, and E) belonging to different organizations (refer to table 1) each following its own software development strategy.

As has been surveyed earlier by other researchers refactoring is not just a phenomenon to smoothen the code but it comes with its own cost and risks [5]. The first team, from Organization A had given a poor response due to the lack of knowledge in the area of refactoring. No one in the team actually was practicing refactoring as they quoted the project being relatively smaller and given the time constraints they could not embark on something less known. Generally a failing code is quick fixed by the developer with his own reasoning and instinct. All the developers were involved in all the tasks and hardly had any specialized person for carrying out task like testing.

The Company D that deals in formulating and delivering website design and development projects uses agile methodology combined with SDLC (system development lifecycle) that helps them ease the whole process. As per the feedback of the developers they had experienced that deferred refactoring sometimes would incur more cost and risk, so the best approach is to follow the test driven development. The team had designated testers for that task.

The company E that works in the domain of Health Care Finance and Enterprise claims to have a good experience in agile development with expert testers for automation testing, manual testers and even unit test writers who assist with the Test Driven Development[8].Agile teams seem to have a better planning in this regard [1].

Table 1: Code Smells and refactorings commonly performed by the software teams

Company Name/Team Size	Technology	Refactoring	Code smell identification Process
A/10	Java	Never refactored or refactored in their own way such as simplifying entagled loop statement with either using recursion or other logic.	Personal experience and knowledge, no specific technique or methodology. For example long methods, long classes, logic duplication, removing unrelated code from a particular method or class
B/17	C#	Rename Method, Rename variable, Rename class, Extract method, Move Method	Manual aswell as automated but none having thorough knowledge about the type of code smells that may be found.
C/15	Java	Extract Method, Move Method ,Rename a variable, a method, a class or even a package name.Inline a method or a variable.	Usually automated refactorings are preferred. The commonly used IDE sare Eclipse and Netbeans.
D/12	C#	Performing manual aswell as automated refactoring depends upon the requirements. Extract class, extract method, extract super class, extract subclass, push down method, pull up method	Code inspection the usual way to perform small cycle refactorings or use a tool like ReSharper,Visual Assist X or JustCode for Visual Studio. Trying to find the smells in places with highest business values.
E/20	.NET Framework	Engage in manual automated refactorings. Use agile development and SCRUM. Generally follow test last approach but a few also follow Test first approach	High level code smells identified by tools. Code rigidity is a bad symptom where there is a lot of dependencies amongst the methods and on other related objects. Configuration data not in a centralized location and scattered through the code reflects bad coding.

IV. MAJOR REASONS ABOUT USING OR NOT USING REFACTORING

The reasons professionals quoted for not refactoring a design can be broadly categorized as follows:

A. Deadlines:

The business pressure to complete the task in a given frame of time on a specific deadline is usually immense on the team as the stakeholders usually decide on the deadlines with the manager without consulting the developers. This also is a big reason for the introduction of technical debts and ironically calls for refactoring in return [16].

It has also been observed that the development teams are not very keen on pursuing the proactive refactoring [4][6] unless there is some business driven requirement or if there is an acute need to refactor because of the deteriorating performance of the product.

B. Lack of Tests

Refactoring without proper testing becomes worthless. Hence if a developer refactors without testing the code there's no way to ensure if the developer has introduced a

bug or has changed the behavior. To maintain the quality in code, some good testers should always be available but that's not the case always in most of the projects. Therefore refactoring without a proper test plan usually takes the developer a week behind on the roadmap without assurance of any improvement.

C. Troublesome and risky.

It has been cited frequently by many authors that to refactor is not an easy task and involves risk in particular introducing new faults or other problems [7][8] and many a times behavior preservation becomes quite difficult especially when inheritance is involved.

D. Technical

Participants reported there are various technical constraints that limit refactoring such as inadequate tool support. Other reasons include the nature of the project that inhibits refactoring. Examples include like working on legacy system that lack test suites, having to implement a third-party interface, non familiarity with the code etc. The professionals working on a legacy system have to put in extra effort to refactor it, especially legacy systems developed in C needs major code rejuvenation practices due to the lack of object oriented constructs, so various techniques have been invented to refactor the code to produce a maintainable and readable code [18][19]. A detailed discussion about the various barriers to refactoring has been highlighted by various researchers already [11][14]. The management support is also a factor as the participants reported that they have to comply with the plan of their boss.

.

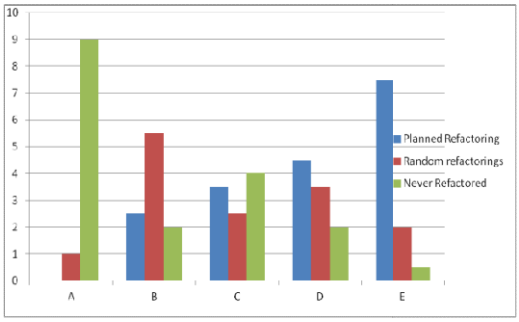

Fig 1: Refactoring pattern in the surveyed organization

E. Analysis of the participants' background:

As indicated in Fig 1, it depicts how the respondents from Company A, B, C, D and E perceive and implement refactoring. As can be seen from the figure the Company A has hardly any candidates those have adopted any planned refactoring strategy and there are many who have no idea about it.

An important observation in this case would be to assess the background of the developers as well. There were almost 5 to 15 % candidates those were fresher or having less than 2 years of experience. These respondents could hardly speak about refactoring. Almost 60 % of the candidates were from the core computing background with bachelors or masters degree and they depicted greater curiosity and awareness. Whereas approximately 30 % of the candidates were from a non computing background such as electronics, electrical or mechanical aswell who found the concept to be difficult to acquire and would not choose to refactor unless it is explicitly required. But for sure the experienced programmers were keen to somehow do away with the technical debt.

V. CONCLUSIONS AND FUTURE WORK

The interactions from five industrial companies explore some conceptions about the refactoring practices in the industry. The goal of this study was to provide insights into the practice of refactoring in software development processes. This questionnaire-based survey performed directly as well as indirectly via email provides results from approximately 80 respondents. The study found that industry still lacks experts in the area and developers in general are reluctant to learn or adopt the skills due to a number of constraints such as time, skills and risk etc. As per the survey results refactoring doesn't seem to impact the overall maintainability or the complexity metrics unless it's performed with a proper layout and plan. Refactoring changes made on a random basis hardly has any impact infact they carry the risk of introducing bugs in the program. The principal results concerning planning and implementation of refactoring indicates 80 % of agile team members do careful planning during the project and have a good idea about refactoring and related tasks like TDD whereas rest of the teams had a relatively lower percentage (approx 20 to 30%) of the participants responding similarly towards careful planning for refactoring. The study is not an exhaustive case as the numbers of participants are small but the projections do indicate the refactoring trend and highlight the gap between research practices and industrial usage. The future work would be directed to a wider participant with broader parameters for assessment.

REFERENCES

[1] Chen, J., Xiao, J., Wang, Q., Leon J. Osterweil., Mingshu L. Empirical Software Eng (2016) 21: 1397. 8 Refactoring planning and practice in agile software development: An empirical study (PDF Download Available).

[2] Roberto Verdecchia, Rene Aparicio Saez , Giuseppe Procaccianti , Patricia Lago (2018) Empirical Evaluation of the Energy Impact of Refactoring Code Smells

[3] 3. Szőke G., Nagy C., Ferenc R., Gyimóthy T. (2014) A Case Study of Refactoring Large-Scale Industrial Systems to Efficiently Improve Source Code Quality. In: Murgante B. et al. (eds) Computational Science and Its Applications – ICCSA 2014. ICCSA 2014. Lecture Notes in Computer Science, vol 8583. Springer, Cham.

[4] 4. Khanam, Zeba and Ahsan, Najeeb Mohd. (2017) Evaluating the Effectiveness of Test Driven Development: Advantages and Pitfalls. In International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 18 (2017) pp. 7705-7716

[5] 5. Kim et al. (2014) An Empirical Study of Refactoring Challenges and Benefits at Microsoft. : IEEE TRANSACTIONS On Software Engineering.

[6] 6. Elssamadisy A, Schalliol G (2002, May). Recognizing and responding to bad smells in extreme programming. In

Proceedings of the 24th International conference on Software Engineering (pp 617–622). ACM

[7] 7. Gabor Szoke, Gabor Antal, Csaba Nagy, Rudolf Ferenc, Tibor Gyimóthy, Empirical study on refactoring large-scale industrial systems and its effects on maintainability, Journal of Systems and Software, Volume 129,
2017, Pages 107-126, ISSN 0164-1212,

[8] Moser R., Abrahamsson P., Pedrycz W., Sillitti A., Succi G. (2008) A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team. In: Meyer B., Nawrocki J.R., Walter B. (eds) Balancing Agility and Formalism in Software Engineering. CEE-SET 2007. Lecture Notes in Computer Science, vol 5082. Springer, Berlin, Heidelberg

[9] Michael W ; Uwe D ; Will S. (2016). Improving Code Maintainability: A Case Study on the Impact of Refactoring. 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)

[10] 10. Rizvi, S., Khanam, Z. (2011) A methodology for refactoring legacy code. In International Conference on Electronics and Computer Technology (ICECT 2011), pp. 198–200 (2011), IEEE Xplore.

[11] Emerson Murphy-Hill., Andrew. (2008) Black Breaking the barriers to successful refactoring. 2008 ACM/IEEE 30th International Conference on Software Engineering

[12] Leppänen, Marko; Mäkinen, Simo; Lahtinen, Samuel; Sievi-Korte, Outi; Tuovinen, Antti-Pekka; Männisto, Tomi, "Refactoring-a Shot in the Dark?," in Software, IEEE, vol.32, no.6, pp.62-70, Nov.-Dec. 2015.

[13] 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 Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on, September 2012, pp. 104–113.

[14] Khanam, Zeba. (2018). Barriers to Refactoring: Issues and Solutions. 2454-4248. 4. 232.

[15] Vidal, Santiago & Marcos, Claudia & Diaz-Pace, Andres. (2014). An approach to prioritize code smells for refactoring. Automated Software Engineering. 10.1007/s10515-014-0175-x.

[16] Marinescu R (2012) Assessing technical debt by identifying design flaws in software systems. IBM Journal of Research and Development 56(5):9

[17] Mkaouer W, Kessentini M, Bechikh S, Cinnéide MÓ, Deb K (2014) Software refactoring under uncertainty: a robust multi-objective approach. In: Genet. Evol. Comput. Conf. GECCO 2014.

[18] Khanam, Z. and Rizvi, S. "Refactoring Catalog for Legacy software using C and Aspect Oriented Language"- the proceedings of SERP 2011, USA.

[19] Z Khanam, SAM Rizvi. "Aspectual Analysis of Legacy Systems: Code Smells and Transformations in C", International Journal of Modern Education and Computer Science, Volume 5, Issue 11, Page 57, 2013.

[20] Chatzigeorgiou, A. and Manakos, A. 2013. Investigating the evolution of code smells in object-oriented systems, Innovations in Systems and Software Engineering. NASA Journal. DOI= 10.1007/s11334-013-0205-z

[21] F. A. Fontana, M. Mangiacavalli, D. Pochiero, and M. Zanoni, "On experimenting refactoring tools to remove code smells," in Scientific Workshop Proceedings of the XP2015. ACM, 25 May 2015, p. 7

[22] Moghadam, I. H. and Ó Cinnéide, M. 2012. Automated Refactoring using Design Differencing, In Proceedings of European Conference on Software Maintenance and Reengineering (ECSM'12).

[23] Rizvi S A M. and Z Khanam. (2010) A Comparative Study of using Object oriented approach and Aspect oriented approach for the Evolution of Legacy System . 2010 International Journal of Computer Applications (0975 – 8887) Volume 1 – No. 7

[24] Harman, M., Mansouri, A. and Zhang, Y. 2012. Search-based software engineering: Trends, techniques and applications. ACM Comput. Surv. 45, 1, Article 11 (December 2012), 61 pages. DOI=10.1145/2379776.2379787.

[25] G. Procaccianti, H. Fernandez, and P. Lago, "Empirical Evaluation of Two Best-Practices for Energy-Efficient Software Development," J. Syst. Softw., vol. 117, no. July 2016, pp. 185–198, 2016

[26] R. Verdecchia, F. Ricchiuti, A. Hankel, P. Lago, and G. Procaccianti, "Green ICT research and challenges," in Advances and New Trends in Environmental Informatics. Springer, 2017, pp. 37–48

[27] J. Koomey, "Growth in data center electricity use 2005 to 2010," A report by Analytical Press, completed at the request of The New York Times, p. 9, 2011.