Journal of the Punjab University Historical Society
Volume: 34, No. 01, January – June 2021

Naveed Jhamat [*]
Zeeshan Arshad [**]
Ghulam Mustafa [***]

Historical Perspective of Software Refactoring Tools towards Future Solution

Abstract

The present study will explore that for the adaptation of software system, we need crucial changes and essential enhancements both at structural and behavioural level. It would also discuss that software refactoring deals with structural refinements in software but keeping its behaviour intact for better understanding, hitch less maintenance and for stifle future changes. For automatic software refactoring numeral approaches and various prototyping tools have been proposed and developed in last couple of decades. Our purpose is to propose an automated software Refactoring tools for future community after critically analyse existing refactoring tools developed in recent and history. At first, we would discuss a Systematic Literature Review to compare scope, methodology and results of various semi and fully automated Refactoring tools proposed in literature for various languages since 2000. Later, as a solution, we would propose a methodological tool based on our findings. In addition to this, we would experiment refactoring tools on open-source software to compare them from different perspective and find out their performance, precision, and recall. It would highlight many opportunities that research community should be familiar with before adding any such complementary automated refactoring tool to list. Our SLR not only underline disparity in the base and outcome of existing tools but also lightens lacking that needs to be eradicated before suggesting any new tools for professional practice.

Introduction

In the late 1980s, (researchers from two distinct Universities: University of Illinois: Bill Opdyke and Ralph Johnson, and from the University of Washington: William G and David Notkin invented Software refactoring approach[1]. Later on, in the early '90s, as a concept, software refactoring was presented by Opdyke in his Ph.D. thesis [2].

[*] *Dr. Naveed Jhamat, Assistant Professor, Department of Information Technology, University of the Punjab, Gujranwala Campus, Pakistan, Email: naveed.jhamat@pugc.edu.pk.*
[**] *Zeeshan Arshad, Lecturer, Department of Information Technology, University of the Punjab, Gujranwala Campus, Pakistan. Email: zeeshan.arshad@pugc.edu.pk*
[***] *Dr. Ghulam Mustafa, Assistant Professor, Department of Information Technology, University of the Punjab, Gujranwala Campus, Pakistan. Email: gmustafa@pugc.edu.pk.*

Software refactoring is defined as the branch of software engineering that deals with internal complexity, maintenance, and other quality attributes of a software. Software refactoring is an approach to transform a written program to enrich its inner structure, design, naivety, understandability, and other characteristics without modifying its external behavior[3]. Software refactoring is one of those troubles that require numerous quality goals to be convinced. The initiative behind software refactoring is to identify variables, methods, and classes to assist future implementations, extensions, and compressive enhancement. Further, this identification is utilized to enrich several software quality aspects such as reusability, maintainability etc.

By the time, the software is developed, enriched, reformed, and added to new prerequisites, the code of the software becomes more complex and quits its original state that results in low-quality software[4]. That is why a significant amount of development funds is devoted to software maintenance. In this regard, better software development approaches and tools are useless to resolve this issue, they are utilized to fulfill more and more requirements of end-users, surrounded by the same time frame they increase the complexity of software.

To tackle this high tendency of complexity, there are such techniques are highly recommended and demanded that cut down the software complexity via incremental enhancement of the internal structure of a software. The domain of research that deals with this subject are indicated as *Software Restructuring* or, particularly in the object-oriented development setting, *Refactoring*[5].

Literature Review:

Refactoring Tools 1997 – 2018: In 1997, researchers from the University of Illinois, Don Roberts ET presented a study in which they addressed the Smalltalk Refactoring browser that performs three main refactorings: (1) Class Variable refactoring, (2) Class refactoring, and (3) method refactoring, automatically and improves internal structure of Smalltalk programs. Later on, in 1999, Shengbing Ren discussed a prototype tool that provides different twenty-four refactoring for the use of case models and visual effects for sketching and showing case models. Since early 2001, IntelliJ IDEA, a refactoring tool supports, a wide array of refactoring for java and only two refactoring implementations: rename and move for other languages. Another Refactoring Tool developed for Java programs Refactor IT, is a commercial tool that detects code smells and performs up to twelve distinct refactoring. It works as a Plugin for the Sun ONE Studio, Eclipse, Oracle, JBuilder, NetBeans IDEs. In 2008, Dmitry Jemerov presented an overview of software refactoring implementation in IntelliJ IDEA and imparted the synopsis of core architectural components of this tool. IntelliJ IDEA is figured among the elementary Java IDEs that crossed the Refactoring Rubicon by Martin Fowler and it provides the multi-lingual refactoring[6]. Another refactoring tool, Deodorant is an Eclipse plugin used to detect Type-Checking bad smells automatically in java source codes and refactors them by the means of implementation of "Replace Conditional with Polymorphism" or "Replace Type Code with state/Strategy" refactoring[7]. In 2010, Miryung Kim presented a study on Ref-Finder, an Eclipse plugin that is based on a template-based refactoring method and performs both atomic as well as composite refactoring automatically. Cristopher Brown focused

on Haskell Refactored known as HaRe and described several new refactoring techniques for the Haskell 98 programs, particularly, discussed structural and data-type refactoring. A study on BeneFactor, an Eclipse plugin, states that Benefactor works with two main components: refactoring detection which runs in the background, and code improvement that modifies existing code to its enhanced form. The BeneFactor refactors Java source codes and is considered among automatic refactoring tools. Katsuhisa Maruyama and Takayuki Omori presented a security-aware software refactoring tool called JSart that is built as a plugin of Eclipse[8]. This tool assists developers to flexibly detect the adverse impact of code modification on the security vulnerabilities and facilitates them to accept or reject implemented refactorings. Iman Hemati Moghadam and Mel Ó Cinnéide discussed Code-Imp, an automated refactoring tool, in their study and their area of focus is the implementation of Code-Imp and the summarization of three major functioning research standards concerning with Code-Imp: "refactoring for testability, metrics exploration, and multi-level design improvements.

From the array of software refactoring tools, *True Refactor* is an automated refactoring tool that extensively enhances the clarity of legacy object-oriented systems and increases the maintainability, reusability, and comprehensibility of legacy software[9]. Asger et al proposed a framework for the specification and implementation of JavaScript refactoring on the utilization of the groundwork of pointer analysis and the key insight of their framework implies that despite the dynamic behavior of JavaScript programming, it is possible to have ultimate appropriateness of JavaScript refactoring utilizing a few queries in the groundwork of pointer analysis.

In 2012, Kwankamol Nongpong introduced a novel semi-automated refactoring tool called *JCodeCanine* that detects bad smells using software metrics within the Java source code and recommends an array of refactoring that assists to enhance the internal attributes of the program[10]. Andreas Thies and Eric Bodden proposed an automated novel and more protective method for the reflective java programs in their study using dynamic program analysis, called *RefaFlex* and they implemented this tool as an open-source Eclipse plugin[11]. Hyrum presented the real-world application of a refactoring tool to refactor large C++ codebases called ClangMR by combining the framework of the Clang compiler and the Map-Reduce parallel processor. Vitor Sales proposed a novel approach for the recommendation of move method refactoring that evaluates the likeliness of dependencies determined by the source method along with the dependencies determined by other methods in objective classes[12]. Wim Mkaouer proposed a novel recommender tool called *DINAR* for software refactoring that adopts refactoring with dynamism and recommends refactoring to programmers interactively according to their feedback and initiated code modifications[13]. Their approach uses NSGA-II to enhance software quality, increase semantic coherence and reduce the number of refactoring, then finds an upfront set of non-dominated refactoring solutions [14].

Thus, the literature review of refactoring reveals a lot of opportunities for the researchers, on the ground of which my motivation and proposed work in based[15]. After critically analyzing literature of Software Refactoring tools, we found that:

1. Accuracy & Performance Issue prevails in many of these refactoring tools.
2. Most tool were tested on smaller and proprietary applications.
3. Most of the tools are language dependent i.e., they refactor software of a single Language only.
4. All available Refactoring opportunities are not supported in any tool.
5. None of the tool deals with the refactoring of all kind of Code Smells. (Mostly tools refactor 4-5 Code Smells out of 22+ known source code Smells).[16]

Sr #	Tool Name	Developed in (Language)	Developed for (Language)	Detection Type (Auto /Semi/ Manual)	Is Plug in (Yes? Plug in of)	Refactor to	Refactor Type	Experimented on (Data)	Precision and Recall (Accuracy)	Approach / Methodology	Working Style
1	RefFinder	Java	Java	Automated	Eclipse	63/72 Fowler's catalog Smells	e.g. Move Method, etc?	(Edit, and revision pairs of Ofoutte and Card.	P: 79% R: 85%	Use of Version History	Two java program versions
2	CodeImp	Java	Java	Automated	No	automated refactoring to improve software quality	Method-Level Refactorings, Field-Level Refactorings		N/A	Use of 25 different quality software metrics	AST representation of Java programs
3	JDeodorant	Java	Java	Automated	Eclipse	automatically identifies Type-Checking bad smells in Java source code	Move Method	Java source code	N/A	type-checking badsmells	
4	CHASE	Monobrian, Gerrit & Code Flow	Code Review		No	Distinguish b/w refactor-ings from non-refactorings during code review	Moved files	open source and industrial software projects	N/A	Distinguish b/w refactor-ings from non-refactorings during code review	renaming variables and breaking large methods into smaller ones.
5	SmellSuite Refactoring Browser	object-Oriented		Automated	No	Improving the structure of Smalltalk programs	move method across object boundary	N/A	N/A	validate the new design separate from the intended functionality.	
6	MORE		A multi-objective search based approach	Automated	No	seven code smells and four design patters	Design patterns eg factory method, visitor, singleton, strategy	popular multi-objective algorithm NSGA-II	84% code smells and average 92%	A multi-objective search based approach	Empirical study based on qualitative and quantitative
7	Rust	N/A	existing infrastructure made available to Rust compiler		No	Restricts duplication of references through borrowing	Move semantics				
8	JAVA SMELL DETECTOR	java	develop IID	automated	No	3/22 java code smells	iterative process	java code		Iterative approach	
9	Icode Cardea	java	java	Semi-Automated	Eclipse plug-in	integrate code smell detector	Extract Method, Move Method	Existing component fluid and Eclipse	Average accuracy 54% without smells	analysis approach	
10	Spartanizer			Automated	Iterative Eclipse plug-in	was invented and fully refactorings to an entire software system		Open source on GitHub		Spartan programming methodology	
11	JDEAR	using a multi-objective evolutionary algorithm NSGA-II.	To improve software quality	Iterative	No	dynamically adapts and suggests refactorings to developers interactively based on their feedback and introduced code changes.	software refactoring dynamically	four large open source systems and one industrial project	N/A	Interactive Recommendation of Refactorings	non-dominated refactoring solutions using NSGA-II
12	Jnari	java	java	Automated	Eclipse plug-in	adverse impact of code changes on security		existing code	N/A	detects downgrading of access level of a field variable within the modified code	
13	RefFlex	java	java	automated	open source Eclipse plug-in	refactorings for reflective Java programs	dynamic program analysis	three open source programs	N/A		Reflection refactoring

Table: List of Software Refactoring Tools

Proposed Methodological Tool

Since 1997 to 2018, plenty of software refactoring tools has been developed. There are several studies available on software refactoring tools that demonstrate different software refactoring tools, approaches and detection methods of code smells (the initial step towards software refactoring[17]. Despite these developments and research since last 2 decades, there is still a list of limitations such as accuracy issue, lack of community trust towards automated refactoring tools[18]. Thus, further enhancements in this field are inevitable and there should be a refactoring tool that improves human trust and facilitate automated refactoring with required accuracy[19].

Refactoring is essentially a step ahead from code smells detection towards their eradication[20]. In our previous work, we after critically analyzing literature, highlighted the limitations of code smells detection tools and techniques. Later on,

we proposed a lightweight code smell detection approach – which is based on the combination of Regex based source code parsing technique and software metrics[21]. Based on that approach we build a prototyping tool "Generic Code Smell Detector - GCSD" which is an (Sparx System) Enterprise Architecture Plug-in and have ability to detect all 22 code smells identified by fowler from multiple languages[22]. After the successful detection of different code smells from the source code of multiple languages, now refactoring of these smelled areas (detected by GCSD) is obligatory[23].

Conclusion

Refactoring is a process of refining and maintaining software code without disturbing its functionality. Literature review on refactoring reveals that a lot of work on refactoring concepts, approaches, tools, and techniques has been done in the past twenty years. Our critical literature review highlighted several limitations in software refactoring tools and we found that accurate software Refactoring is heavily based on the appropriate detection of code bad smells. To cope up with limitations in existing software refactoring tools we will extend our prototyping tool i.e., Generic Code Smell Detection (GCSD) which is based on code smell detection. GCSD tool will have the ability to not only detect code smells from multiple languages but also suggest and apply the suitable refactoring to automatically correct these smells from the source code. We will test our tool on various projects developed in multiple languages such as Java, Python, C# to measure its effectiveness and accuracy for the community.

Notes & References

[1] Dig, M., Griswold Danny, William G., Emerson Murphy-Hill, and Schäfer. 2014. "The Future of Refactoring (Dagstuhl Seminar 14211." Drops. Dagstuhl. De 4: 40–67,

[2] Opdyke, W. F. (1992). Refactoring object-oriented frameworks.

[3] Abebe, M., and C. Yoo. 2014. "Classification and Summarization of Software Refactoring Research: A Literature Review Approach."

[4] Murphy-Hill, Emerson. 2006. "Improving Usability of Refactoring Tools." In Companion to the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications - OOPSLA '06. New York, New York, USA: ACM Press.

[5] Mkaouer, Mohamed Wiem, Marouane Kessentini, Slim Bechikh, Mel Ó Cinnéide, and Kalyanmoy Deb. 2016. "On the Use of Many Quality Attributes for Software Refactoring: A Many-Objective Search-Based Software Engineering Approach." Empirical Software Engineer 21, no. 6: 2503–45.

[6] Mens, T., and T. Tourwe. 2004. "A Survey of Software Refactoring." IEEE Transactions on Software Engineering 30, no. 2: 126–39.

[7] Lientz, B. P., and E. B. Swanson. 1980. "Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations." Softw. Maint. Manag. a Study Maint. Comput. Appl. Softw 4: 294,

[8] Guimaraes, Tor. 1983. "Managing Application Program Maintenance Expenditures." Communications of the ACM 26, no. 10: 739–46.

[9] Coleman, D., D. Ash, B. Lowther, and P. Oman. 1994. "Using Metrics to Evaluate Software System Maintainability." Computer 27, no. 8: 44–49.

[10] Glass, R. L. 1998. "Maintenance: Less Is Not More." IEEE Software 15, no. 4: 67–68.

[11] Griswold, William G., and David Notkin. 1993. "Automated Assistance for Program Restructuring." ACM Transactions on Software Engineering and Methodology 2, no. 3: 228–69.

[12] Arnold, R. S. 1986. "An Introduction to Software Restructuring." Tutor. Softw. Restrict.

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

[14] Opdyke, W. F. 1992. "Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks."

[15] Rasool, Ghulam, and Zeeshan Arshad. 2015. "A Review of Code Smell Mining Techniques: Code Smell Mining Techniques." Journal of Software (Malden, MA) 27, no. 11: 867–95.

[16] Roberts, Don, John Brant, and Ralph Johnson. 1997. "A Refactoring Tool for Smalltalk." Theory and Practice of Object Systems 3, no. 4: 253–63.

[17] Ren, S., G. Butler, K. Rui, J. Xu, W. Yu, and R. Luo. 1999. "A Prototype Tool for Use Case Refactoring." Inf. Syst, 173–178,

[18] Jemerov, Dmitry. 2008. "Implementing Refactorings in IntelliJ IDEA." In Proceedings of the 2nd Workshop on Refactoring Tools - WRT '08. New York, New York, USA: ACM Press.

[19] Nongpong, K., and J. T. Boyland. 2012. "Integrating 'Code Smells' Detection with Refactoring Tool Support."

[20] Gil, Yossi, and Matteo Orru. 2017. "The Spartanizer: Massive Automatic Refactoring." In 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE.

[21]Mealy, E., and P. Strooper. 2006. "Evaluating Software Refactoring Tool Support." In Australian Software Engineering Conference (ASWEC'06). IEEE.

[22] Mumtaz, Haris, Mohammad Alshayeb, Sajjad Mahmood, and Mahmood Niazi. 2018. "An Empirical Study to Improve Software Security through the Application of Code Refactoring." Information and Software Technology 96: 112–25.

[23] Tsantalis, Nikolaos, Theodoros Chaikalis, and Alexander Chatzigeorgiou. 2008. "JDeodorant: Identification and Removal of Type-Checking Bad Smells." In 2008 12th European Conference on Software Maintenance and Reengineering. IEEE.