Mining and Managing Big Data Refactoring for Design Improvement: Are We There Yet?

Eman Abdullah AlOmar¹, Mohamed Wiem Mkaouer¹, and Ali Ouni²

 ¹ Rochester Institute of Technology, Rochester, New York, USA
 ² ETS Montreal, University of Quebec, Montreal, QC, Canada

Abstract. Refactoring is a set of code changes applied to improve the internal structure of a program, without altering its external behavior. With the rise of continuous integration and the awareness of the necessity of managing technical debt, refactoring has become even more popular in recent software builds. Recent studies indicate that developers often perform refactorings. If we consider all refactorings performed across all projects, this consists of the refactoring knowledge that represents a rich source of information that can be useful for both developers and practitioners to better understand how refactoring is being applied in practice. However, mining, processing, and extracting useful insights, from this plethora of refactorings, seems to be challenging. In this book chapter, we take a dive into how refactoring can be mined and preprocessed. We discuss all design concepts and structural metrics that can also be mined along with refactoring operations to understand their impact better. We further investigate the many practical challenges for such extraction. The volume, velocity, and variety of extracted data require careful planning. We outline the appropriate techniques from a large number of available technologies for such system implementation.

 ${\bf Keywords:} \ {\rm refactoring, \ software \ maintenance, \ software \ quality}$

1 Introduction

Successful software systems undergo evolution through the continuous code changes, as means to update features, fix bugs, and produce a more reliable and efficient product. Prior studies have pointed out how software complexity can be a serious obstacle preventing the ease of software evolution, as large and sophisticated modules are, in general, harder to understand, and error-prone. Such patterns, located in the system design, negatively impact the overall quality of software as they are responsible for making its design inadequate for evolution. In this context, it has been shown that software engineers spend up to 60% of their programming time in reading source code, and trying to understand its functionality, in order to properly perform the needed changes without "breaking" the code. Consequently, software maintenance activities that are related to improving the overall software quality to take up to 67% of the cost allocated for

the project. The *de-facto* way of handling such debt is through software refactoring. By definition, refactoring is the art of improving design structure while preserving the overall external behavior. With the rise of technical debt, and developers acknowledgment of shortage in their deliverables, refactoring stands as a critical task to maintain the existence of software and to prevent it from decay.

Projects that are known to be successful in maintaining their quality through several waves of updates and migrations across various programming paradigms and frameworks, are known to be witnessing efficient refactoring strategies. Such hidden knowledge has triggered the intention of research to mine and understand how developers refactor their code in practice. In this context, several refactoring detection tools have been lately proposed to mine the development history of a given software project, and extract all the information related to all refactoring operations that were performed on its code elements.

As recent refactoring tools (e.g., RefactoringMiner [24] and RefDiff [21]) have reached a high level of maturity, their usage across various large projects has triggered an explosion in the information that can be obtained regarding previously performed refactorings, and their corresponding impact on the source code. Furthermore, refactoring, being by nature a code change, when batched, becomes harder to analyze. Moreover, code changes visualization is gaining more attention in software engineering research, yet visualizing refactoring is still underresearched.

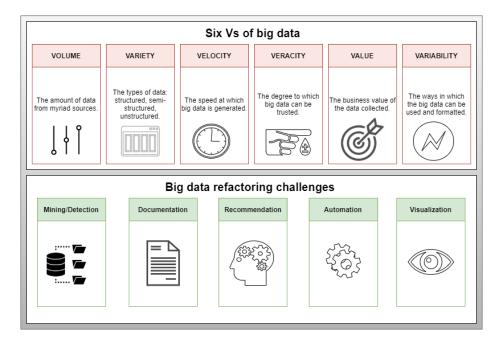


Fig. 1: Big data refactoring

For the above mentioned challenges that the plethora of refactorings have emerged, this chapter initiates the discussion about how the world of big data can provide a rich source of solutions. We detail the multiple challenges linked to refactoring indexing, analysis and visualization, while exploring potential big data solutions. As depicted in Figure 1, we identify five refactoring challenges, triggering the explosion of refactoring data, which we can call *Big Data Refactoring Challenges*. These challenges are 1) Detection of refactoring activities, 3) Recommendation of refactoring opportunities on existing software systems, 5) Automation of refactoring execution, and 5) Visualization of refactoring impact on the source code. We organize this chapter to explore each of these challenges, by detailing its existing tools and methodologies, along with discussing their limitations and how they are explicitly or implicitly linked to big data dimensions.

This chapter is organized as follows: the first section is associated with tools and techniques related to the identification of executed refactorings, the next section is dedicated for documentation. Section 4 summarizes the existing tools to automate the generation of refactorings. Recommending refactorings is also covered in Section 5. The need for refactoring visualization is covered in Section 6 before concluding in Section 7.

2 Mining and Detection

The popularity of the GitHub hosting service is increasing rapidly and has been used frequently for the base of data collection in literature. Research in mining software repositories mainly relies on two GitHub services: the version and bug tracking systems. GitHub stores all versions of the source code and any specific changes are represented by a commit that involves a textual description of the change (i.e., commit message). The bug tracking system, on the other hand, provides an interface for reporting errors. GitHub makes it possible to mine a large amount of information and different properties of open source projects.

The challenge in this area lies in analyzing a comprehensive and large number of GitHub commits containing refactoring. Several studies have mining tools to identify refactoring operations between two versions of a software system. Dig et al. [9] developed a tool called Refactoring Crawler, which uses syntax and graph analysis to detect refactorings. Prete et al. [20] proposed Ref-Finder, which identifies complex refactorings using a template-based approach. Hayashi et al. [12] considered the detection of refactorings as a search problem. The authors proposed a graph search algorithm to model changes between software versions. Xing and Stroulia [27] proposed JDevAn, which is a UMLDiff based, design-level analyzer for detecting refactorings in the history of Object-Oriented systems. Tsantalis et al. presented Refactorings within Git commits. Silva and Valente [21] extended RefactoringMiner by combining the heuristics-based static analysis with code similarity (TF-IDF weighting scheme) to identify 13 refactoring types. Tsantalis et al. [24] extended their tool to enhance the accuracy of

4 Eman Abdullah AlOmar, Mohamed Wiem Mkaouer, and Ali Ouni

the 28 refactoring types that can be detected through structural constraints. A recent survey by Tan [22] compares several refactoring detection tools and shows that RefactoringMiner is currently the most accurate refactoring detection tool. The choice of the mining tool is driven by accuracy; therefore RefactoringMiner is suitable for mining and detecting refactorings and extracting big data refactoring. It is suitable for studies that require a large variety of repositories and commit volumes.

Item	Count
Studied projects	3,795
Commits with refactorings	$322,\!479$
Refactoring operations	$1,\!208,\!970$
Commits with refactorings & Keywords	2,312
Remove false positive commits	1,067
Final dataset	1,245

Table 1: Studied dataset statistics.

With the existence of millions of software projects, whose sizes vary from small to large, mining their refactorings could lead to an amount of data that cannot be handled by traditional means. This links mining refactoring to Big Data's Volume. For instance, in our recent study [2], we mined refactoring in 3,795 open source projects. The process extracted over 1,200,000 refactoring operations, distributed in 322,479 commits. More details about this dataset is in Table 1. We faced challenges in hosting and querying this data. To extend our study, we need to extract refactorings in over 300 000 open source projects, and we are currently unable to perform this study, unless we seek the right framework to collect, store, and index such data.

Another interesting challenge related to such data, is its heterogeneity. Refactoring operations are different from each other in their structure, target code elements and impact on source code. For instance, the rename identifier refactoring, is the act of changing the name of a given attribute. Such operation requires saving the old name of the attribute, its new name and the path of the file containing the attribute. As for extract method, which is the splitting of a given method into two sub-methods, this operation requires saving the old method signature, and body (and path) along with saving the signature and bodies of the newly created methods (and paths). So, each refactoring type requires a unique structure to store its information. Furthermore, various studies are interested in the reachability of the refactoring operation, to better analyze their impact on the code design. Storing refactored code elements and their corresponding dependencies may require specific data structures like graphs. For large and complex systems, analyzing such information is challenging.

3 Refactoring Documentation

A number of studies have focused recently on the identification and detection of refactoring activities during the software life cycle. One of the common approaches to identify refactoring activities is to analyze the commit messages in version-controlled repositories. Prior work [2] has explored how developers document their refactoring activities in commit messages; this activity is called Self-Admitted Refactoring or Self-Affirmed Refactoring (SAR). In particular, SAR refers to the situation which shows developers explicit documentation of refactoring operations intentionally introduced during a code change. For example, by manual inspection of the Cassandra-unit1 open source project, AlOmar et al [2] used this example to demonstrate SAR: "refactoring of Abstract*DataSet to delete duplicate code," which indicates that developers intentionally refactor one class to remove the redundancy antipattern that violates design principles. The authors manually analyzed commit messages by reading through 58,131 commits. Then they extracted, from these commit messages, a set of repetitive keywords and phrases which are specific to refactoring. They provided a set of 87 patterns, identified across 3,795 open source projects. Since this approach heavily depends on the manual inspection of commit messages, in follow-up work, AlOmar et al. [3] presented a two-step approach that firstly distinguishes whether a commit message potentially contains an explicit description of a refactoring effort. Then, secondly classifies it into one of the three common categories identified in previous study [2], which is the first attempt to automate the detection and classification of self-affirmed refactorings. The existence of such patterns unlocks more studies that question the developers' perception of quality attributes (e.g., coupling, complexity); these results may be used to recommend future refactoring activity. For instance, AlOmar et al. [4] identified which quality models are more in line with the developer's vision of quality optimization when they explicitly mention in the commit messages that they refactor to improve these quality attributes. This study shows that, although there is a variety of structural metrics can represent internal quality attributes, not all of them can measure what developers consider to be an improvement in their source code. Based on their empirical investigation, for metrics that are associated with quality attributes, there are different degrees of improvement and degradation of software quality for different SAR patterns.

As stated above, developers use a variety of patterns to express their refactoring activities. Previous studies illustrate such a pattern. However, one big challenge is that it is not practical for large real world projects to manually collect all potential keywords/phrases reported in a large number of commit messages, as developers may use various expressions to annotate how they refactor. To cope with this challenge, future research could plan to use the findings of previous studies to build a text-mining tool that will automatically support software engineers in the task of identifying, detecting, and highlighting selfaffirmed refactoring in the commit messages. This detector could allow users to train their own model and integrate self-affirmed refactoring detectors into their development tools.

Indep of a point	Potential Candidate Refactoring Patterns						
<table-row><table-row><table-row><table-row><table-row></table-row><table-row><table-row></table-row><table-row><table-row></table-row></table-row></table-row></table-row></table-row></table-row></table-row>	BugFix				Internal		
<table-row><table-row><table-row><table-row><table-row><table-row></table-row><table-row><table-row></table-row><table-row><table-row><table-row><table-row></table-row><table-row><table-row></table-row></table-row></table-row></table-row></table-row></table-row></table-row></table-row></table-row></table-row></table-row>							
<table-row><table-row><table-row><table-row></table-row><table-row><table-row></table-row><table-row></table-row><table-row></table-row><table-row></table-row><table-row></table-row></table-row></table-row></table-row></table-row>	10		Improv* code reuse	Add new feature			
Carctor of your control of your inf your 	-						
ConceptNote with a part of the sector of the s	Bug hunting	Code duplication removed	Increased flexibility	New module			
Inder the start of the star	Correction of bug	Delet* duplicate code	More flexibility	Fix some GUI			
Fir offer minister affairer andel et al.Allet more stackingAllet more stacking<	Improv* error handling		Provide flexibility	Added interesting feature	Reduced the code size		
Find any bugFind adjuicts methodBetter excludingAlong facture to suportRelace completyFin summark bugPiller deglineRelace adjuict codeAlong so facturesRelace adjuict codeFind an mol bugRelace adjuict codeCode readbilly quinticationAlong so facturesRelace adjuict codeFind an mol bugRelace adjuict codeEnder readbillyFeature solidRelace adjuict codeFind an mol bugRelace adjuict codeMale is batter readbillyImplement code factureAlong adjuict codeFind any bugRelace adjuict codeMale is batter readbillyImplement code factureAlong adjuict codeFind any bugRelace adjuict codeRelaceRelaceRelaceRelaceFind any bugRelace adjuict codeRelaceRelaceRelaceRelaceFind adjuict codeRelaceRelaceRelaceRelaceRelaceFind adjuict codeRelaceRelacities adjuict codeRelaceRelaceFind adjuict codeRelacities adjuint codeRelaceRelaceRelaceFind adjuict codeRelacities adjuint codeRelaceRelaceRelaceFind adjuict codeRelacities adjuint codeRelaceRelaceRelaceFind adjuict codeSchelling relaceRelaceRelaceRelaceFind adjuict codeRelaceRelaceRelaceRelaceFind adjuict codeRelaceRelaceRelaceRelaceFind adjuict codeRelaceRelaceRelaceRela	Fix further thread safety issues		A hit more readable	Added more features	Complexity has been reduced		
Fin moren longFin depictsBatter maching ProceedingsAlong for fatureAlong for fatureAlong for fatureAlong for fatureAlong for fatureAlong for fature along ProceedingsAlong for fature along for fature ProceedingsAlong for fature <td></td> <td></td> <td></td> <td></td> <td></td>							
Fix availabilityAddition of any of any duplication of the company of present of any of a			Better readability and testabil-				
InclumineNormal stateNormal stateNormal stateNormal stateFind a micro upReduce at ond code dupleExater radabilityFeature duplemation of fatureAdde hubrinanceFind analy upReduced dupleate codeNormal reduced dupleate codeNormal reduced dupleate codeAdde hubrinanceFind analy upClear up and dupleateNormal reduced dupleate codeNormal reduced dupleate codeNormal reduced dupleate codeNormal reduced dupleate codeFind analy upClear up and dupleateNormal reduced dupleate codeReduckily reduced dupleate codeNormal reduced dupleate codeNormal reduced dupleate codeFind analy upClear up and dupleateNormal reduced dupleate codeReduckily reduced dupleate codeNormal reduced dupleate codeNormal reduced dupleate codeFind analy upRemore commented codeReduckily reduced dupleateRemore commented codeReduckily reduced dupleateRemore commented codeFind word nameRemore some code dupleateSourif reduced factorRemore some code dupleateSourif reduced factorNormal source dupleateFind word nameRemore some code dupleateSourif reduced factorNormal source dupleateNormal source dupleateNormal source dupleateFind word nameRemore some code dupleateAdd factor code dupleateNormal source dupleateNormal source dupleateFind word nameRemore some code dupleateAdd factor code dupleateNormal source dupleateNormal source dupleateFind word nameRemore some code dupleateAdd factor code duplea	F	1.1.1. I. Parto de L					
ionionionletryFird at tridy upReface code dupCatemarger ereababilityImplementation of fastureAbled support to inheritance and fast some bayesAbled interpretation of fastureAbled interpretation of fasture <td>Fix several bug</td> <td>Joining duplicate code</td> <td>Code readability optimization</td> <td>Addition of a new feature</td> <td></td>	Fix several bug	Joining duplicate code	Code readability optimization	Addition of a new feature			
Fied and by ogeReface of deplacionmove readablyinformation of fatureAdd import to the inhof- fature fature for a start of a start	Fixed a minor bug		Easier readability	Feature added			
intermal baseIndexFunctionFunctionFunctionContract readability information of fautureAnd chargeport to the information fautureFind analy baseClass up a small deign the termMale in their smallable informationHinder and the information of fauture fautureAnd chargeport to the information fautureAnd chargeport to the information fautureFind analy baseClass up a small deign the termMale in their smallable informationHinder and the information fautureHinder and the information fautureClass up a small deign the informationFind and the informationRemove concentered out con fautureRemove concentered out con fautureRemove fauture fautureRemove fauture<	Final a trialm hum		Immunia naadahilitu				
Fix-suppleRelated out repetionIncrease readablyInformation of fatureAdda to port on the inder information or pairwardAdda to port on the inder information or pairwardAdd	Fixed a tricky bug	Reduce code duplication	Improve readability		Add inneritance		
Field and bying Refactored depicate code Mail in better analable frate and bying in the sequentional in a parameter of the sequentional in a parameter of the sequention in a sequention in the sequention is the sequenticon is the sequent sequentis the sequenticon is the sequention is t	Fix* small bug	Reduced code repetition	Increase readability				
Fix* some bag* Fix* some tage Fix some minor bag* Fix some minor bag* 	Fixed nasty bug	Refactored duplicate code	Make it better readable	Implemented the experimental			
InclusionInstructFaturesInstructInstru							
Field some more bagTempos/Intelling has based on the formate of a subsidiary and supportability Remove the default factorsChange" interfactor text by port of extend and supportability Remove the default factorsChange" interfactor text by port of extend and supportability Remove the default factorsChange" interfactor text by port of extend and support extend text by port of extend and support extend factor and support extend and support e	Fix* some bug*	Clear up a small design flaw	Make it more readable				
basis Renove commende out code provement Resubability and supportability provement Remove incomplete faiture (incomplete faiture) Extend the generated classe (incomplete faiture) Fix* grow* Removed is los of code dupits Removed incomplete faiture (incomplete faiture) Removed incomplete faiture) Fix* issue* Removed faiture dupits Simplify reactability (incomplete faiture) Simplify reactability (incomplete faiture) Simplify reactability (incomplete faiture) Simplify reactability (incomplete faiture) Removed incomplete faiture) Removed incomplete (incomplete faiture)	Fixed some minor bugs	TemporalField has been refac-	Readability enhancement				
Interpretation Important interpretation Important interpretation Important interpretation Important interpretation The suppretation Reachable important interpretation Reacoult interpretation Reacoult interpretation Reacoult interpretation The incorrect Reacoult interpretation Reacoult interpretation Reacoult interpretation Reacoult interpretation Reacoult interpretation The incorrect interpretation Reacoult interpretation Reacoult interpretation Reacoult interpretation Reacoult interpretation Reacoult interpretation The incorrect interpretation Reacoult interpretation Stability improvement Adde functionality interpretation Adde functionality interpretation Reacoult interpretation Stoled some minor bus Reacoult digitation Reacoult interpretation Adde functionality interpretation Adde functionality interpretation Reacoult interpretation Stoled some minor bus Reacoult interpretation Reacoult interpretation Adde functionality interpretation Reacoult interpretation A first the insure Reacoult interpretation Reacoult interpretation Reacoult interpretation Reacoult interpretation Reacoult insuppretation					-		
Fix* topo* Remove a lot of code dupi. Readowally improvement. Remove dup control dupi. Fix* torsken Remove code dupication Reformated for readability Remove frame one of the fixer. Remove interface Fix* torsken Remove some ode dupication Simplif readability Remove frame one of the fixer. Remove interface Fix* soue* Remove some ode dupication Simplif readability Remove frame one of the fixer. Remove frame one of the fixer. Fix* concurrency issue* with Remove dupicate code Scalability improvement Adde functionality Adde functionality Removed housing interface Fixe* concurrency issue* Remove dupicate code Better performance Adde functionality Consolitationality for merge Adde functionality Removed housing interface Solved some minor bug Seri or formance Removed merge and dupicate More mange and Consolitationality Consolitationality Removed housing interface Ref to the issue Removed apparently dupicate More mange and Consolitationality Separate consolitationality Separate consolitationality Separate consolitationality Remove apparentity dupication and the consolitation Remove ap	bugfix*	Remove commented out code		Remove the default feature			
cickiecetexie <th< td=""><td>Fix* typo*</td><td>Removed a lot of code dupli-</td><td></td><td>Removed incomplete features</td><td></td></th<>	Fix* typo*	Removed a lot of code dupli-		Removed incomplete features			
Fix-incorrect Remove some code duplication Simplify readability Remanded some infortance for consistency. Remove the big code duplic ratio Improve "stability Remanded some infortance for consistency. Remanded some infortance for consistency. Infortance for consistency. Infortance for consistency. Infortance for consistency. Remanded duplicate code plaction Add Parformance for consistence. Support of optional feature consistence. Added functionality for consistence. Easter consistence. A for to fissise Remove disc code Remove afficace. Remove for code and disc. Remove		cation			tance		
ionionfor outsistencyfor consistencyfor consistencyFix* surver lissure additionRemoved some dead addUpdate the performanceSupport of optional fasture additionInderivatione managementFix* several issuesResolved duplicate codeAdd "performanceSupport of optional fastureAddit functionality for mergeAddit metrological inderivationeSolved some minor bugsResolved duplicate codeSclability imporvementAdde functionality for mergeAddit functionality for functionality fo							
Fix* sever Removed the big code dupli. Imporve* testability Small feature addition Inheritance management Fix* several issue* Removed some dead and duplicate code. Update the performance Support of optional feature Lower demoved transmission Fix* several issues Remové duplicate code. Scalability improvement Adde functionality Prevents coleral inheritance Solved some more bug Remové duplicate code. Scalability improvement Adde functionality Compatibility that inheritance Solved some more bug Remové dead code Improve "nemethe inheritance Compatibility that inheritance Support of optionality Compatibility that inheritance Solved some more bug Remove duplicated field Hage performance Adde functionality Compatibility that inheritance A to to is some Removed all dead code More efficient* Export functionality Better performance Inhis is a bit of dead code Increase fiftency Extend functionality Better performance Inhis is a bit of dead code Increase fiftency Extend functionality Better performance Inhis is a bit of dead code Increase fiftency Extend functionality Better enapublation	Fix [™] incorrect		Simplify readability		Remove inheritance		
cationcationcationUpdate the parformanceSupport of optional formLocate the medule dependenceFit* several issuesResolved digitate codeAdd* performanceSupport of optional foreived facPresenticaliar inheritanceWorking on a bugRenoved digitate codeSubbilly improvementAdded functionality orPresenticaliar inheritanceWorking on a bugRenove digitated fieldHuge performanceAdded functionality orComposition better than inheritanceGet rid ofRenove digitated fieldHuge performanceAdds two new piecesComposition better than inheritanceGet rid ofRenove dad codeImprove performanceAdds two new piecesSeparates concernsA his of a simple solution toRenove apparently dataMale it reusable for otherEpsert functionalityMales polymorphinFix a colle pointRenove apparently dataIncrease efficiencyExtend functionalityRenove dependencyFix a colle pointFix* code smallIncrease efficiencyExtend functionalityRenove dependencyFix* anor subsFix* code smallIncrease performance improve mentFunctionalityRenove dependencyRenov* apparentJudate dataseLargel resolutionFunctionalityRenove dependencyRenov* apparentJudate dataseLargel resolutionFunctionalityRenove dependencyRenov* apparentSale performance improve mentHuge performance improve mentHuge performance improve mentRenove dependencyRenov* apparentRenove data dat	Fix* issue*	Removed the big code dupli-	Improve* testability		Inheritance management		
networkplicate codenetworkNetwork </td <td></td> <td></td> <td></td> <td></td> <td></td>							
Fix* concurrency issue* with Remov* duplicate code Removed issues Add* performance Removed issues Supporting for derived feasibility improvement Supporting for derived feasibility improvement Addef functionality Add using interfance Added functionality for merge Solved some minor bag Sort to horendous code using interfance Addef functionality for merge Add composition Working on bag Remov* duplicate field Hige performance improve merge Adds functionality for merge Add composition Get rid of Remov* data code Improv* performance Adds to new pines of functionality for merge Separates concerns A fix to the issue Remove add add code More manageable Consolviton Separates concerns Fix a couple of issue Remove add code Increase efficiency Extend functionality Better encapabilition and less diverses Fix* anion rissue Fix* code smell Increase efficiency Extend functionality Better encapabilition and less diverses Additional fixes Fix* code smell Increase efficiency Extend functionality Better encapabilition and less diverses Additional fixes Fix* code smell Increase efficiency Event functionality Better encapabilition and less diverse Additional fixes Fix* code smell Increase efficiency Event functionality Panove diverse Fix* finite	⊢ix* several issue*		Update the performance	Support of optional feature			
Fires sevent issues Resolved duplicate code Stability important Added functionality or maps of the sevent hormadous code du plication Better performance ment Added functionality for maps Adde composition Working on a bug Removed publicated field Huge performance mprove ment Adde functionality for maps Adde sone new pieces Composition better than in- herance As to the issue Removed add code More afficient Disciplination Separates concerns To to the issue Removed add code More afficient Disciplination Separates concerns To a simple solution to the issue Removed add code More afficient Disciplination Separates concerns To a souple of issue Removed add code Increase afficients Export functionality of Better encapsulation Better encapsulation Fix* minor issue Remove and more dead code Increase afficiency Extend functionality Better encapsulation Correct* sue Remove some code small Large performance improve ment Increase afficiency More afficiencionality Parter consultation Remove some code small Large performance improve ment House common functionality Pareov* adegraded upici	Fix* concurrency issue* with		Add* performance	Supporting for derived fea-			
Solved some minor bugs piction Sort out horeselus code du- get idon Better performance ment Addef functionality for merge Addis rue functionality composition better than in- heritance Get id of A bit of a simple solution to the issue Remove duplated field ment Huge performance ment Adds two new pieces of func- bianity to consultate common func- bianity. U* composition to consultate common func- bianity. U* composition to consultate common func- bianity. U* composition to consultate common func- bianity. Development of this function. Developmentor this function.			-				
InteractionplicationHuge performance improve mentAdding mer functionality ment interactComposition better than in- metracceGet rid of Get rid of Consolidate common func- the issueRemove dead codeImprove performance tonality toAdding the functionality tonality toComposition to Consolidate common func- tonalityA bit of a simple solution to the issueRemoved apparently dead consolidate common func- tonalityRemoved apparently dead consolidate common func- alityBetter Analiting of polymor plannFix a couple of issue rissueRemoved apparently dead consolidate interviewMake it resusable for other bit for its in bit of dead codeIncrease efficiencyExtent functionality Better encapulation Better encapulation Better encapulation Better encapulation dependenciesFix minor issueFix* some code smell merstsIncrease efficiencyFunctionality addedBetter encapulation dependenciesRemove apparently dead concet smellFix* some code smell increase performance merstsHouse common functionality issueRemove dependenciesCorrect test failure*Update data classes tak code smellLos of performance merstMove functionality issueSplit out each module into issueFix* and period fix* complie failureAutipattern bad for perfor mance faulty codeMore functionality issueRemoved issue period intro alityFix* and merstLos of merformance merstPulling-up common functionality issuePulling-up common functionality issueFix* and merst <td< td=""><td></td><td></td><td></td><td>,</td><td></td></td<>				,			
Cert do Remov [*] dead code Improve [*] performance to all y to consolidate common func- tionality to Device on func- tionality to Device on consolidate common func- consolidate common func- consolidate common func- consolidate common func- consolidate common func- code Separate concerns Fix a couple of issue Fix more separate common func- code Removed apparently dead Make it reusable for other end to code Extent functionality of Extra to common functionality Removed more dead code Performance ment Extra to common functionality Removed dependencies Additional fixes fixe come set is send remot Fixe 'some code smell Increase efficiency ment Extra to common functionality Removed dependencies Additional fixes fixe complie failure * Remove dependencies Maxie performance improve- ment Nove functionality Removed deviced y unant susce Remove dependencies Fixe complie failure * Update data classes failuty code Loss of performance ment Nove functionality Removed deviced y unant susce Nove functionality Removed inductionality Split out each module into susce Fixe annel failure task in dimension func- faulty code Paling-up common functionality Removed genements Paling-up common functionality Removed genements Palind went into Removed genements Fixe complie failure traily old common funct	Solved some minor bugs		Better performance	Added functionality for merge	Add composition		
Get rid of Remov* dead code Improv* performance Adds two new pieces of functionality U* composition A bit of a simple solution to Remove some dead-code More manageable Consolidate common functionality Separates concerns A fin to the issue Removed all dad code More efficient* Development of this functionality Batter handling of polymorphism Fix a couple of issue Removed and ead code Increase efficiency Extend functionality Makes polymorphism easier State management This is a bit of dead code Verify correctness Extract common functionality Better encapulation of legendencies Correct issue Fix* code smell Increase efficiency House common functionality Remov* dependency Additional fixes Fix* action code smells' Largely resolved performance Improve functionality Split out each module into issue Fix* all failure * Update data classes Lists of performance Improve functionality Split out each module into issue Fix* and failure test failure* Update data classe Lists of performance Improve functionality Split out each module into issue Fix* complit failure Removet	Working on a bug	Remove duplicated field	Huge performance improve-	Adding new functionality			
Ab id a simple solution to the issueRemove some dead-codeMore manageable consolityConsolity common func simple solution to tionalitySeparates concernsA fix to the issueRemoved all dead codeMore efficient*Development of this function altyBetter handling of polymor- philmFix a couple of issueRemoved apparently dead codeMake it reusable for other exase efficiencyExtend functionality of Extend functionality of Better encapsulation and less dependenciesFix* minor issueRemove dead codeVerify correctnessExtract common functionality issueBetter encapsulation and less dependenciesCorrect issueFix* code smellIncrease efficiencyExtend functionality issuesPubled down dependenciesRemove more some code smellIncrease erformance issuesHuse common functionality issuesRemove dependencyResolv "poblemRemove some code smellsIncrease erformance issuesHuse common functionality issuesSplit out each module into issuesResolv "poblemRemove useless classMassing the performance mentMove functionality issuesSplit out each module into issuesFix* ornipit failureRemove duseless classMassing the performance improvements provementsMove functionality issuesMove functionality issuesRet or nor miscage Less of modifications to code toyleUsability improvements provements provements provementsMove functionality issuesRet or mersage handing Fix* some cordAntipattern hasd for perfor mance improvements	6	D *1 1 1					
A bit of a simple solution to the issue Remove and dead-code More anageable Consolidate common func- tionality Separates concerns tionality A fix to the issue Removed all dead code More afficient* Development of this funcion ality Better handling of polymor- hism Fix a couple of issue Removed apparently dead code Make it reusable for other code Export functionality Better encapsulation Fix* minor issue Removed more dead code Increase efficiency Extend functionality of demendencies Better encapsulation and less dependencies Correct issue Fix* some code smells Largely resolved performance ment House common functionality and the intro- issue Pushed down dependencies Correct* test failure* Update data classes Lots of performance improve- ment House common functionality subity improvement More functionality more functionality Pushed down dependencies Fix* annel failut etx* Remove duviously und/- faulty code Update data classes Lots of performance improve- ment More functionality Pushed functionality Fix* annel failut etx* Remove duviously und/- faulty code Update data classes Optimizing the speed Reside factoring issing function- ality Pushed functionality Fix* corral Less ong methods	Get rid of	Remov* dead code	Improv* performance		Us* composition		
A fix to the issue Removed al dead code More efficient* Development of this function alty Better handling of polymor- alty Fix a couple of issue Removed apparently dead Make it reusable for other code Expert functionality Makes polymorphism easier Issue management This is bit of dead code Increase efficiency Extend functionality of extent common functionality Better encapsulation Gorrect issue Fix* doed smell Increase performance ment House common functionality alwaise performance improve- ment Functionality added Pushed down dependencies Gorrect issue Fix* some code smells' issues Largely resolved performance ment House common functionality splt Pushed down dependencies Gorrect issue Update data classes Lots of performance improve- ment Measuring the performance ment House common functionality Permove alwais Fix* annihe failure Update data classes Lots of performance improve- ment More efficients Pushed founctionality Fix* comple failure Anipattern bad for perfor- mances Pushed founctionality Removed alwaise performance improve- ent issue Pushed founctionality Exter error hunding Anipattern bad for perfor- mances Performance boot Refactoring existing function- ality Pushed founctionality Fix* errora* Removed some unncessary fix* errora* <td< td=""><td>A bit of a simple solution to</td><td>Remove some dead-code</td><td>More manageable</td><td>,</td><td>Separates concerns</td></td<>	A bit of a simple solution to	Remove some dead-code	More manageable	,	Separates concerns		
Image: second					2		
Fix a couple of issue Removel apparently deal code Make it reusable for other Export functionality Makes polymorphism easier code Issue management This is a bit of dead code Increase efficiency Extend functionality of the encoupluation Better encapsulation and less dependencies Correct issue Fix* code smell Massive performance improve- iment Functionality added Pushed down dependencies Additional fixes Fix* some code smell Increase performance House common functionality Split out each module into issue Recolv* problem Remove* some code smell Largely resolved performance Mouse functionality Split out each module into issue Fix* all failed text* Remove useless class Measuring the performance Moved shared functionality Split out each module into issue Fix* and failed text* Removed useless class Measuring the performance Moved shared functionality Split out each module into issue Fix* for the errors Lots of modifications to code style Usability improvements Pulling-up common functionality Better encore additional interact intervences and fields Performance improvement Reimprovement Better encore additional intercose performa	A fix to the issue	Removed all dead code	More efficient*				
Isue mangementThis is a bit of dead codeIncrease efficiencyExtract functionality of Better encapsulationBetter encapsulation appendenciasFix* minor issueRemoved more dead codeVerify correctnessExtract common functionality addedBetter encapsulation appendenciasCorrect issueFix* code smellIncrease performance mentIncrease performanceHouse common functionality split out each module into issuesAdditional fixesFix* some code smellIncrease performance issuesHouse common functionality issuesRemov* dependenciasResolv problemRemov* some code smells issuesLots of performance improve- mentMove functionalitySplit out each module intoFix* all failed test*Remove useless classMessaring the performance mentMoved shared functionalitySplit out each module intoFix* comple failure failury codeRemoved sobiously unused/ tyleImprox* stabilityFature/code improvementsSplit out each module intoBetter encry shalling mancesArtipattern bad for perfor mancesNoticable performance improvement nancesPubling-up common functionalityBetter encry shalling tripleAntipattern bad for performance boostRefactored functionalityImprox*Better encry shalling tripleRemoved some uncessary feldsPerformance tobostRefactored functionalityFix* some enco*Less long methodsPerformance endorement alityRemove functionalityFix* some encosImprox*Refactor performance etal menceRemove functional	Fix a couple of issue	Removed apparently dead	Make it reusable for other	,	Provide and the second s		
Fix* minor issue Removed more dead code Verify correctness Extract common functionality dependencies. Better encapsulation and less dependencies. Correct issue Fix* code smell Massive performance improve ment Functionality added Pushed down dependencies. Additional fixes Fix* some code smell Increase performance House common functionality Remov* dependency Correct test failure* Update data classes Loss of performance improve ment Move functionality Split out each module into issues Correct test failure* Remove useless class Messuring the performance into soft performance Move functionality Split out each module into issues Fix* and failed test* Remove useless class Messuring the performance into soft performance Move functionality Improved functionality Fix* comple failure Removed boxiously unsed/- faulty code Massing performance into soft performance Move functionality Improved functionality Exter error handling Attipattern bad for perfor- faulty code Optimizing the speed Refactoring existing function- tions Improved functionality Exter error message Less long methods Performance onprovement Add functionality with Improved functionality Fix* some error* Less long methods Performance test Performance speed-up Refactoring existing function- tions					-		
Correct issueFix* code smellMasive performance improve mentIncrease performanceHouse common functionality mentPeuhed down dependenciesAdditional fixesFix* some code smellLargely resolved performance issuesHouse common functionalityRemov* dependencyResolv* problemRemov* some code smellsLargely resolved performance issuesMove functionalityRemov* dependenciesCorrect* test failure*Update data classesMeasuring the performance mentMove functionalityFix* code improvementsFix* all failed test*Remove useless classMeasuring the performance mentMoved shared functionalityFix* code improvementsFix* corne imply codeAntipattern bad for perfor mancesPublicy up romono functionalityFix* code improvementsPublicy up romono functionalityBetter error handling mancesAntipattern bad for perfor mancesOptimizing the speed mentReinplemented missing func- tionality andFix* code improvementFix* oronLess long methodsPerformance enproved mentRefactored functionalityFix* codeFix* oron*Removed some unnecessary Performance coptimizationRefactoring scisting function- tion*Fix* codeFix* some error*Less long methodsPerformance tepped- Performance tepped- tion*Removing dasoleter functionalityFix* some error*Removed some unnecessary Performance tepped- tion*Refactoring scisting function- tion*Fix* some error*Removed some unnecessary performance tent performance tent performance tent							
nentnentnentnentAdditional firesFix* some code smellsIncrease performance lasuesImproved functionalitySplit out each module into lasuesResolv* problemRemove data classesLos of performance improve mentMove functionalitySplit out each module into lasuesCorrect* test failure*Update data classesMeasuring the performance mentMove functionalityImprovements alityFix* all failed test*Remove useless classMeasuring the performance mentMove functionalityImprovements alityFix* compile failureRemoved obviously unued/- failuty codeImprove * tabilityFature/ code improvements alityImprovements alityBetter error handlingAntipatern bad for perfor- mancesNoticeable performance im- provementPush more functionalityImprovements alityCleanup error messageLess long methodsPerformance boostRefactored functionality oImprovements factored functionality oFix* error*Removed some unnecessary feldsPerformance optimizationMerging its functionality toImprovements factored functionality oFix* some error*ImprovementsRefactore performance test frest some error*Refactore performance test for formance speed-upRemoved functionalityFix test errorImprovementsRefactore performance test fix some error*Removed some unnecessary function- its waller errorImprovementsFix test errorImprovementsRefactore performance test for formance testRemoved functionali	Fix* minor issue	Removed more dead code	verity correctness	Extract common functionality			
Additional fixes Fix* some code smell Increase performance House common functionality Remov* dependency Resol** problem Remov* some 'code smells' Largely resolved performance Improved functionality Split out each module into Correct* test failure* Update data classes Lots of performance improve Move functionality Feature/code improvements Fix* and failed test* Remove dovisory unused/. Improv * faublity improv * stability Feature/code improvements Improvements A fix for the errors Lots of modifications to code Usability improvements Puling-up common function- ality Improvement Better error handling Antipattern bad for perfor Noticable performance boost Re-implemented missing func- tions Remove dovisor Crow fix* Removed some unnecessary Performance enhancement fields Refactoring existing function- ality Improved functionality Fix* error* Performance specify Remove support Improved functionality Fix some errors Refactor performance test Removire functionality Fix some errors Refactor performance test Removire functionality	Correct issue	Fix* code smell		Functionality added	Pushed down dependencies		
Resolv* problem Remov* some 'code smells' issues Largely resolved performance issues Improved functionality issues Split out each module into issues Correct* test failure* Update data classes Lots of performance improve ment Move functionality Improved functionality Fix* all failed test* Remove useless class Measuring the performance Move functionality Improved functionality Fix* failure* Remove dobiously unused/- failuty code Improv* stability Feature/code improvements ality Polling-up common function- ality Better error handling Antipattern bad for perfor- mances Noticeable performance to provement Push more functionality Improve stability Better error message Less long methods Performance boost Refactored functionality to fields Remove functionality Fix* some error* Remove functionalization Merging its functionality with Improvement Fix* some error* Performance peed-up fields Remove functionality Improvement Fix* some error* Performance speed-up fields Remove functionality Improvement Fix* some error* Remove functionality Remove functionality Improvemance beed-up fiel	A 1 P.1 1 C	 • •			D * 1 - 1		
Correct* test failure*Update data classes mentLots of performance improve mentMove functionalityFix* all failed test*Remove obviously unused/ faulty codeImprov* stabilityFeature/code improvementsA fix for the errorsLots of modifications to code styleUsability improvementsPulling-up common function- alityBetter error handlingAntipattern bad for perfor- mancesNoticeable performance im- provementPush more functionalityBetter error message handlingKilled really old commentsOptimizing the speedRe-implemented missing func- tionsCleanup error messageLess long methodsPerformance in- performance enhancementRefactoring existing functionalityFixed worgError*Performance improvementAdd functionalityFixed worgPerformance improvementAdd functionality toFixed worgRemoved and some uncecesary performance speed-up ris* some error*Removed sobolet functionalityFix some errorsRefactor performance test Resource and performance test rormane and performance test rormane during obsolete functionalityFix test errorSpeed up performance compatible with spit out the GUI function rityFixed more compilation errorsSpeed to performance test Resource and performance test resource and performance test resource compilation errorsFixed some compilation errorsSpeed up performance compatible with spit out the GUI function fixed more compilation errorsFixed more compilation errorsFix backward compatible with spit migning ratio compatible with spit mus				,			
nemtmentmentFix* all failutes*Removue doviously unused/. faulty codeMeasuring the performanceMoved shared functionalityFix* compile failureRemoved obviously unused/. faulty codeImprov* stabilityFeature/code improvementsA fix for the errorsLots of modifications to code styleUsability improvementsPulling-up common function- ality.Better error handlingAntipattern bad for perfor- mancesNoticeable performance im- provementPush more functionalityBetter error messageLess long methodsPerformance boostRe-implemented missing functionsCleanup error messageLess long methodsPerformance improvementAdd functionalityFixed wrongRemoved some unnecessaryPerformance improvementAdd functionality toFix* error*Performance improvementAdd functionality toFix* some errorsRemoved some unnecessaryPerformance speed-up Removed functionality thiFix some errorsRemoved some performance testRemoved functionalityFix compile errorRemade performance testRemoved functionalityFix test errorRemade performanceRemoved compatibilityFixed some compilation errorsSpeed up performanceReplicating existing function- alityFixed some compilation errorsRemoved functionalityRemovedFixed some errorsRefactor performance testRemoved functionalityFix some errorsRemoved functionalityAdd supportFixed some errorsRest functionalityAdd support </td <td>resort problem</td> <td>itemov some code smens</td> <td></td> <td>Improved functionality</td> <td>opine oue cuen module meo</td>	resort problem	itemov some code smens		Improved functionality	opine oue cuen module meo		
Fix* all failed test* Remove useless class Measuring the performance Moved shared functionality Fix* compile failure Removed obviously unused/, lamprov* stability Fature/code improvements Pulling-up common function- ality A fix for the errors Lots of modifications to code style Usability improvements Pulling-up common function- ality Better error handling Antipattern bad for perfor- mances Noticeable performance im- provement Push more functionality Better error message handling Killed really old comments Optimizing the speed Re-implemented missing func- tions Cleanup error message Less long methods Performance boost Refactoring existing function- ality Fixed wrong Performance improvement Add functionality to Improvement Fix* some error* Performance endonization Remove functionality with Improvement Fix* some errors Refactor performance test Remove functionality Improvement Fix compile error Refactor performance test Removing ubsolete functionality Improvement Fix test error Speed up performance Replicating existing function- ality Improvement Replicating existing function- ality Improvereting functionality	Correct* test failure*	Update data classes		Move functionality			
Fix* compile failure faulty code Removed obviously unused/, faulty code Improv* stability faulty code Feature/code improvements A fix for the errors Lots of modifications to code style Usability improvements Pulling-up common function- ality Better error handling Antipattern bad for perfor- mances Noticeable performance im- provement Push more functionality Better error message handling Killed really old comments Optimizing the speed Re-implemented missing func- tions Cleanup error message Less long methods Performance boost Refactoring existing function- ality Fixed wrong Removed some unnecessary fields Performance much improveent Add functionality to Fix* arors* Performance coptimization Merging its functionality Improver Fix some errors Refactor performance test fix compile error Refactor performance test fix compile error Resource functionality Fix terror Speed up performance fixed some compilation errors Speed up performance test fixed some compilation errors Replicating existing function- ality Fixed some compilation errors Fix backward compatiblity Add cosmetic changes Existing function- ality Fixed some compilation errors	Fix* all failed test*	Remove useless class		Moved shared functionality			
faulty codeIndexIndexIndexIndexA fix for the errorsLots of modifications to code styleUsability improvementsPulling-up common function- alityBetter error handlingAntipattern bad for perfor mancesNoticeable performance im- provementPulling-up common functionalityBetter error message handlingKilled really old commentsOptimizing the speed feiting the speedRefactored functionalityCleanup error messageLess long methodsPerformance boostRefactoring existing function- alityFixed wrongError fix*Removed some unnecessary fieldsPerformance much improvementAdd functionality toFixet arror*Performance optimizationMerging its functionalityMerging its functionalityFix* some error*Performance optimizationMerging its functionalityFix some errorsRefactor performance espeed-up rorsRemove* unnecessary func- tion*Fix test errorSpeed up performance test Fix backward compatible withReplicating existing function- alityFix test errorSpeed up performance Fix backward compatiblityAdd cosmetic changesFixed almore compilation errorsSpeed up performance Fix backward compatiblityAdd cosmetic changesFixed almore error*PerformanceReplicating existing function- alityFixed almore errorSpit out the GUI functionFixed almore errorFix backward compatiblityAdd cosmetic changesFixed almore errorPerformanceReplicating existing function- alityFi							
styleattractalityBetter error handlingAntipattern bad for performance im provementPush more functionalityBetter error message handlingKilled really old commentsOptimizing the speedRe-implemented missing func- tionsCleanup error messageLess long methodsPerformance boostRefactored functionalityError fix*Removed some unnecessary fieldsPerformance improvementAdd functionality toFixed wongImplemented missing function- alityImplemented missing function*Fix* some error*Performance optimizationMerging its functionality withFix* some error*Performance optimizationMerging its functionalityFix some errorsImplemented missing function*Implemented missing function*Fix test errorImplementedRenamed performance testRemoving obsolet functionalityFixe all build errorsImplementedSplet out the GUI functionImplementedFixed function errorsImplementedFixe kormation compatibilityAdd* supportFixed failing testsImplementedFixed compatibilityAdd* supportFixed rearImplementedMake it compatibilityAdd* supportFixed rearImplementedMake it compatibilityAdd* supportFixed rear <td< td=""><td></td><td>faulty code</td><td></td><td>· ·</td><td></td></td<>		faulty code		· ·			
Better error handling Antipattern bad for performances improvement Push more functionality Better error message handling Killed really old comments Optimizing the speed Re-implemented missing functionality Cleanup error message Less long methods Performance boost Refactored functionality Error fix* Removed some unnecessary fields Performance enhancement Refactoring existing function- ality Fix# orror* Removed some unnecessary fields Performance optimization Merging its functionality with Fix* some error* Performance speed-up Remove functionality Performance speed-up Fix some errors Refactor performance test Reworke functionality Fix compile error Renamed performance test Reworke functionality Fixe all or compilation errors Backward compatible with Split out the GUI function ality with Fixes all build errors Fix backward compatible with Split out the GUI function Fixes all build errors Fully compatible with Implement* new architecture Handling tests Fully compatible with Implement* new architecture Handle Keep backwards compatible GUI shanle changed Fix* arrow* More compatible	A fix for the errors		Usability improvements				
nancesprovementneckBetter error message handlingKilled really old commentsOptimizing the speedRe-implemented missing func- tionsCleanup error messageLess long methodsPerformance boostRefactored functionalityError fix*Removed some unnecessary fieldsPerformance enhancementRefactored functionality toFixed wrongCPerformance much improvementAdd functionality toFix* error*Image: Commance much improvementAdd functionality toFix* some error*Image: Commance much improvedRemove* function*Fix some error*Performance much improvedRemove* function*Fix some errorsImage: Commance much improvedRemove* unnecessary func- tion*Fix compile errorRenoreRelactor performance testRemovig doslete functional ityFix ter errorSpeed up performanceReplicating existing function- ality withFixed more compilation errorsImage: Commandia testSplit out the GUI functionFixed some compilation errorsFixing function- singration compatibilityAdd's supportFixed Failing testsImage: Commandia testReplicating existing function- ality withFixed ror*Image: Commandia testReplicating existing function- ality withFixed some compilatio errorsImage: Commandia testSplit out the GUI functionFixed some compilatio errorsImage: Commandia testSplit out the GUI functionFixed some compilatior errorsImage: Commandia testSplit out the GUI function	Better error handling		Noticeable performance im-				
Cleanup error messageLess long methodsPerformance boostRefactored functionalityCleanup error messageLess long methodsPerformance enhancementRefactored functionalityFixed wrongRemoved some unnecessaryPerformance improvementAdd functionality toFixed wrongPerformance much improvedRemove* function*Fix* error*Performance optimizationMerging its functionality withFix* some error*Performance speed-upRemove* unnecessary func- tion*Fix some errorsRemove function*Remove* unnecessary func- tion*Fix compile errorRefactor performance testReworked functionalityFix dome errorsRenamed performance testRemoving obsolete functional- ity withFix compile errorSpeed up performanceReplitating existing function- ality withFixed some compilation errorsBackward compatible withSplit out the GUI functionFixed some compile errorFix backward compatible withMold commet: changesFixed shill errorsFully compatible withImplement* new architectureFixed shill errorsMate it compatible withUl abdateHandleAddMore compatible withUl sout has changedFix* error*Mate it compatible withUl abdateFixed shill errorsRetains error handlingAdd's supportFixes all build errorsFully compatible withUl abdateFixes all build errorsRetains compatible withUl abdateFixes all build errorsRetains backward compatibleUl		mances					
Cleanup error message Less long methods Performance boost Refactoring existing functionality Error fix* Removed some unnecessary fields Performance mhancement Refactoring existing function- ality Fixed wrong Performance improvement Add functionality to Fix* error* Performance much improved Remov* function* Fix some errors Performance speed-up Remov* function* Fix some errors Refactor performance test Removing obsolet functionality Fix compile error Refactor performance test Removing obsolet functionality Fix test error Speed up performance Replicating existing function- ality with Fixed wore compile errors Backward compatible with Split out the GUI function Fixed some compile errors Fixing migration compatiblity Add "support Fixed all build errors Fixing migration compatiblity Add "support Fixed all compatible with Maintain compatiblity Add itonal changes Fixed some compile error* Maintain compatible with Update Fixed soling error* Maintain compatible with Update Fixed some compile erros*	Better error message handling	Killed really old comments	Optimizing the speed				
Error fix* Removed some unnecessary fields Performance enhancement ality Refactoring existing function- ality Fixed wrong Performance improvement Add functionality to Fix* error* Performance optimization Merging its functionality with Fix* some error* Performance optimization Merging its functionality with Fix some errors Refactor performance test Reworke functionality Fix compile error Refactor performance test Reworke functionality Fix test error Speed up performance Replicating existing function- ality with Fixed more compilation errors Backward compatible with Split out the GUI function Fixes all build errors Fix backward compatible with Split out the GUI function Fixes all build errors Fully compatible with Implement* new architecture Handle Keep backwards compatible Update Handling error* More compatible with UI shout Trow* fix* Should be backward Compatible Weit Support Fixes all build errors Mark it compatible with UI layout has changed Fix* problem* More compatible <td>Cleanup error message</td> <td>Less long methods</td> <td>Performance boost</td> <td></td> <td></td>	Cleanup error message	Less long methods	Performance boost				
fieldsIntroductFixed wrongPerformance improvementAdd functionality toFixed wrongPerformance optimizationMerging its functionality withFix* some error*Performance optimizationMerging its functionality withFix some errorsPerformance speed-upRemove* unnecessary func- tion*Fix some errorsPerformance testReworke functionalityFix compile errorRenamed performance testReworke functionalityFix test errorRenamed performanceReplicating existing function- ality withFixed more compilation errorsBackward compatible withSplit out the GUI functionFixed Failing testsFully compatible withMedicomaticationFixed Failing testsFully compatible withMole compatible withHandleKeep backwards compatiblityAdd cosmetic changesHandle error*Mait i compatible withUl alyout has changedError* fix*Mait i compatible withUl alyout has changedTwashig error handlingMore compatible withGUI: Small changesFix* problem*Should be backward compatible withUl shout has changedFix* problem*Retains backward compatibleNew Ul layoutGot rid of deprecated codeAddesStay compatible withPelet* deprecated codeId AddesId changesFix* problem*AddesIl changesFix* problem*AddesIl changesFix* problem*AddesIl changesFix* problem*AddesIl changes <td< td=""><td></td><td>Removed some unnecessary</td><td></td><td>Refactoring existing function-</td><td></td></td<>		Removed some unnecessary		Refactoring existing function-			
Fix* error* Performance much improved Remov* function* Fix* some error* Performance optimization Merging its functionality with Fix some errors Performance optimization Merging its functionality with Fix some errors Refactor performance test Reworked functionality Fix compile error Renamed performance test Reworked functionality Fix test error Speed up performance Replicating existing function- ality with Fixe and performance Replicating existing function- ality with Performance Fixed some compile errors Backward compatible with Split out the GUI function Fixes all build errors Fix backward compatible with Split out the GUI function Fixes all build errors Fixing migration compatibility Add cosmetic changes Fixes all build errors Fully compatible with Implement* new architecture Handling More compatibility Add iconal-totanges Firx* fix* Make it compatible with Ul Jayout has changed Tore* fix* Should be backward Compatible Fix* problem* Retains backward compatibility Various fix* Got rid of deprecated code Stay compatible with Ul enhancements Delet* deprecated code Added some robustness Encoresting function- ality with		fields					
Fix* some error* Performance optimization Merging its functionality with Fix some errors Performance speed-up Remove* unnecessary func- tion* Fix some errors Refactor performance test Reworked functionality Fix compile error Renamed performance test Removing obsolete functionality Fix test error Speed up performance Replicating existing function- ality with Fixed more compilation errors Backward compatible with Split out the GUI function Fixed some compile errors Fix backward compatibility Add' support Fixed Failing tests Fully compatible with Implement* new architecture Handle Keep backwards compatibility Add' support Handle error* Mairt in compatibility Add' support Error* fix* Make it compatible with Ul layout has changed Yarious fix* Should be backward compatible GUI: Small changes Fix* problem* Retains backward compatible Ul shout fix* problem* Retains backward compatible Ul changes Fix* problem Adde some robustness New Ul layout							
Fix small error Performance speed-up Remove* unnecessary func- tion* Fix some errors Refactor performance test Removide functionality Fix compile error Renamed performance test Removide functionality Fix test error Speed up performance Replicating existing function- ality with Fixet error Backward compatibility Add cosmetic changes Fixed some compile errors Backward compatibility Add * support Fixed some compile errors Fixing migration compatibility Add* support Fixed some compile errors Fixing migration compatibility Add* support Fixed some compile errors Fixing migration compatibility Add* support Fixed some compile errors Maintain compatibility Add* support Fixed some compile errors Maintain compatibility Add tosmetic changes Fixed some compile errors Maintain compatibility Additional changes for Handle Keep backward compatibile with Ul ayout has changed Horis fix* Make it compatible with Ul ayout has changed Various fix* Should be backward New Ul layout Should be backward Compatible New Ul ayout Got rid of deprecated code Stay compatible with Ul enhancements Delet* deprecated code							
Fix some errors Refactor performance test Reworked functionality Fix compile error Renamed performance test Removing obsolet functionality Fix test error Speed up performance Replicating existing function- ality with Fix test error Speed up performance Replicating existing function- ality with Fixed more compilation errors Backward compatible with Split out the GUI function Fixed some compile errors Fix backward compatibility Add* support Fixed sall build errors Fixing migration compatibility Add* support Fixed Failing tests Method be backwards compatibile Update Handle Keep backwards compatibility Additional changes for Handling error* Make it compatible with Ul alyout has changed Various fix* Should be backward compatibility Additional changes Fix* problem* Retains backward compatibility Id changes Fix* problem*				Remove* unnecessary func-			
Fix compile error Renamed performance test ity Removing obsolete functional- ity Fix test error Speed up performance Replicating existing function- ality with Fixed some compile errors Backward compatibility Add cosmetic changes Fixed some compile errors Fixing migration compatibility Add cosmetic changes Fixed some compile errors Fixing migration compatibility Add * support Fixed some compile errors Fixing migration compatibility Add * support Fixed some compile errors Fixing migration compatibility Add * support Fixed some compile errors Fixing migration compatibility Add * support Fixed some compile errors Maintain compatibility Add tional changes Handle Keep backwards compatibile Update Handling error* Make it compatibile with Ul ayout has changed Tweaking error handling More compatibile GUI: Small changes Various fix* Should be backward compatible New Ul layout Fix* problem* Retains backward compatibile Ul changes Fix* problem* Retains backward compatible Ul enhancements Delet* deprecated code Added some robustness E Remov* deprecated code Improve robustness E <td></td> <td></td> <td></td> <td></td> <td></td>							
IndexIndexityItyFix test errorSpeed up performanceReplicating existing function- ality withReplicating existing function- ality withFixed more compilation errorsBackward compatible withSplit out the CUI functionFixed some compile errorsFix backward compatible withSplit out the CUI functionFixes all build errorsFixing migration compatibilityAdd cosmetic changesFixes all build errorsFully compatible withImplement* new architectureHandleKeep backwards compatibilityAdd fixinal changes forHandle gror*Mate it compatibile withUl Jayout has changedFixer fix*Make it compatible withUl Jayout has changedTreashing error handlingMore compatibleCUI Small changesVarious fix*Should be backward compatibleNew Ul JayoutFix* problem*Retains backward compatibleUl changesFix* deprecated codeStay compatible withUl enhancementsDelet* deprecated codeAdded some robustnessImprove robustness							
Fix test error Speed up performance Replicating existing function- ality with Fixed more compilation errors Backward compatible with Split out the CUI function Fixed some compile errors Fix backward compatibility Add cosmetic changes Fixed sall build errors Fixing migration compatibility Add* support Fixed Failing tests Fully compatible with Implement* new architecture Handle Keep backwards compatibility Add* support Handle Keep backwards compatibility Add* support Firs* Maintain compatibile with Update Handle Mee it compatible with Ul Jayout has changed For* firs* Make it compatible with Ul Jayout has changed Various fix* Should be backward compatible GUI: Small changes Fix* problem* Retains backward compatible Ul changes for rid of deprecated code Stay compatible with Ul enhancements Delet* deprecated code Added some robustness Emover dupteres Remov* deprecated code Improve robustness Emover dupteres	r ix complic ciror		renamed performance test				
Fixed more compilation errors Backward compatible with Split out the GUI function Fixed some compile errors Fix backward compatibility Add cosmetic changes Fixes all build errors Fixing migration compatibility Add support Fixed Failing tests Fully compatible with Implement* new architecture Handle Keep backwards compatibility Add "support Handle error* Maintain compatibility Additional changes for Error* fix* Make it compatible with UI layout has changed Various fix* Should be backward Compatible Fix* problem* Retains backward compatibility VI changes Fix* problem* Retains backward compatibility UI enhancements Delet* deprecated code Added some robustness Errore for endertine some robustness	Fix test error		Speed up performance	Replicating existing function-			
Fixed some compile errors Fix backward compatibility Add cosmetic changes Fixes all build errors Fixing migration compatibility Add* support Fixed Failing tests Fully compatible with Implement* new architecture Handle Keep backwards compatibility Add* support Handle Keep backwards compatibile Update Handle Keep backwards compatibility Additional changes for Error* fix* Make it compatible with Ul layout has changed Tweaking error handling More compatible GUI: Small changes Various fix* Should be backward compatibility New UI layout Fix* problem* Retains backward compatibility Ul changes Got rid of deprecated code Stay compatible with UI enhancements Delet* deprecated code Improve robustness Emprove robustness	Fixed more compilation arrest		Bachward compatible with				
Fixes all build errors Fixing migration compatibility Add* support Fixed Failing tests Fully compatible with Implement* new architecture Handle Keep backwards compatible Update Handling error* Maintain compatibility Additional changes for Error* fix* Make it compatible Ul ayout has changed Various fix* Should be backward compatible GUI: Small changes Fix* problem* Retains backward compatible Iu changes Got rid of deprecated code Stay compatible with Ul enhancements Delet* deprecated code Added some robustness Emov* deprecated code							
Fixed Failing tests Fully compatible with Implement* new architecture Handle Keep backwards compatible Update Handle geror* Maintain compatibility Additional changes for Error* fix* Make it compatible with Ul layout has changed Tweaking error handling More compatible GUI: Small changes Various fix* Should be backward compatible New UI layout Fix* problem* Retains backward compatible ity Ul changes Got rid of deprecated code Stay compatible with UI enhancements Delet* deprecated code Improve robustness Metodes for							
Handling error* Maintain compatibility Additional changes for Error* fix* Make it compatible with UI layout has changed Tweaking error handling More compatible GUI: Small changes Various fix* Should be backward compatible New UI layout Fix* problem* Retains backward compatible with UI changes Got rid of deprecated code Stay compatible with UI enhancements Delet* deprecated code Improve robustness Emens*	Fixed Failing tests		Fully compatible with	Implement* new architecture			
Error* fix* Make it compatible with Ul Jayout has changed Tweaking error handling More compatible GUI: Small changes Various fix* Should be backward compatible New UI Jayout Fix* problem* Retains backward compatible ity UI enhancements Got rid of deprecated code Stay compatible with UI enhancements Plet* deprecated code Added some robustness Encompatible							
Tweaking error handling More compatible GUI: Small changes Various fix* Should be backward compatible New UI layout Fix* problem* Retains backward compatibile ity Ul changes Got rid of deprecated code Stay compatible with UI enhancements Delet* deprecated code Added some robustness E Remov* deprecated code Improve robustness E							
Various fix* Should be backward compatible New UI layout compatible Fix* problem* Retains backward compatible ity UI changes Got rid of deprecated code Stay compatible with UI enhancements Delet* deprecated code Added some robustness Improve robustness Remov* deprecated code Improve robustness Improve robustness							
Image: comparisible Image: comparisible Image: comparisible Fix* problem* Retains backward compatible ity UI changes Got rid of deprecated code Stay compatible with UI enhancements Delet* deprecated code Added some robustness Improve robustness			Should be backward-				
ity ity Got rid of deprecated code Stay compatible with UI enhancements Delet* deprecated code Added some robustness enhancements Remov* deprecated code Improve robustness enhancements							
Got rid of deprecated code Stay compatible with UI enhancements Delet* deprecated code Added some robustness Remov* deprecated code Improve robustness	Fix [*] problem [*]			UI changes			
Delet* deprecated code Added some robustness	Got rid of deprecated code			UI enhancements			
	Delet* deprecated code		Added some robustness				
Improve usability	Remov* deprecated code						

Table 2: Potential candidate refactoring text patterns.

If we want to extend the study of AlOmar et al. [4], and analyze refactoring documentation across the dataset previously described in Table 1, we are challenged by the Volume of text that needs to be analyzed. Furthermore, this text is originated from many developers, from different projects, and so, it contains various semantics, which increases the ambiguity of deciphering it. From a Variability perspective, there is a need to find better formatting and indexing for this text in order to adequately extract the needed information. For instance, the rise of word2Vec [11], when combined with the appropriate vector indexing, may provide a potential solution to avoid naive string matching, which is known to generate false positives. Other topic modeling techniques can be also explored to extract textual patterns which are relevant to refactoring documentation, however, their manual validation is challenging due to the large number of potential patterns that can be generated. For instance, Table 2, showcases the existence of various refactoring candidate textual patterns, extracted from our dataset in Table 1, and which require manual validation.

4 Refactoring Automation

Maintaining large scale code and ensuring large scale semantically safe refactoring can be a challenging task. Many contemporary IDEs provide a limited set of automatic refactoring operations applied to a single file or package. Handling large refactoring poses a big challenge in many object-oriented development projects. Further, performing a high volume of refactoring typically takes longer and changes multiple parts of the system. If refactoring influences large chunks of the system, as a result, there is a need to break changes down into smaller parts. A few questions could be investigated when performing Volume and Variety of refactoring:

- How can large refactoring operations be planned?
- How can undo-functionality be implemented for large refactorings during the actual refactoring?
- How can we proceed to add more functionality during the execution of large refactorings while ensuring behavior preservation for the existing application?
- How can we integrate the plans of implementing large refactorings into the development process?
- How can we document the status of a large refactoring?

4.1 Refactoring Tools

Various aspects of refactoring need to be considered when automating the application of refactoring. These include, but are not limited to, automation, reliability, coverage, and scalability of refactoring tools. With regards to automation, fully automated and semi-automated refactoring tools are beneficial for developers. For example, adding support of an "undo" feature can facilitate the process 8 Eman Abdullah AlOmar, Mohamed Wiem Mkaouer, and Ali Ouni

of returning the software to its original state in case the effect of refactoring is not desirable. Reliability indicates whether the software guarantees behavior preservation of the refactoring transformation. A full guarantee of behavioral preservation is challenging, thus, an automated refactoring tool should define a set of pre and post conditions to ensure program correctness after the application of refactoring. Concerning coverage, refactoring tools should cover a wide range of refactoring activities that developers could perform, i.e., the tool should be as complete as possible. It would be worthwhile to have refactoring tools that support a complete set of refactoring operations of different levels of granularity (e.g., class, method, package) to improve the system design from different perspectives (e.g., code smell removal, adherence to object-oriented design practices such as SOLID and GRASP, etc). Scalability is another aspect that should be taken into consideration when constructing refactoring tools.

4.2 Lack of Use

Despite the positive aspects of semi-automated refactoring, many developers continue to prefer to do refactoring manually, even when the opportunity to use a refactoring tool presents itself. In the realm of Extract Method refactoring, Kim et al. [13] found that 58.3% of developers chose to perform their refactorings manually. Another study by Negara et al. [18] shows that even though the majority of developers aware of refactoring tools and their benefits, they still chose to refactor manually. Murphy-Hill et al. [16] found that only 2 out of 16 students in an object-oriented programming class had previously used refactoring tools. Another survey by Murphy-Hill [15] found that 63% of surveyed individuals at an Agile methodology conference used environments with refactoring tools, and that they use the tools 68% of the time when one is available. This is significant, since Agile methodologies are generally predisposed to be in favor of refactoring, indicating the general usage must be even lower. Murphy-Hill tempers this statement by noting the likelihood of bias in the participants' responses, as well as the survey size of 112 being non-representative as it is comparatively small compared to all programmers.

Murphy-Hill also compared studies by Murphy-Hill et al. and Mäntylä et al. They show that students claim they are more likely to perform Extract Method refactoring immediately compared to Rename refactoring, yet developers are 8 times as likely to use a Rename refactoring tool than an Extract Method refactoring tool [14]. Research by Vakilian et al. and Kim et al. also indicate that the majority of developers would prefer to apply refactorings other than Rename refactoring manually [25, 13]. There is no clear conclusion for this discrepancy, but it indicates either an underuse of Extract Method refactoring tools or overuse of Rename refactoring tools. Ultimately, it seems unrealistic to come to a concrete conclusion regarding the use of refactoring tools by all developers, but these findings show strong indirect evidence that refactoring tools are underutilized compared to their potential.

From big data perspective, these studies suffer from lack of analysis of Value. There should be an alignment of how tools refactor code with what developers are expecting their code to be refactored. So far, existing tools focus on removing code smells, and improving the design structural measurements, however, and as seen in Table 2, developers do refactor their code for various reasons that go beyond these two objectives.

4.3 Lack of Trust

There have been a number of studies and surveys done collecting information on developers' aversion to refactoring tools. Surveys by Campbell et al. [8], Pinto et al. [19], and Murphy-Hill [15] include the same barrier to entry in their findings: lack of trust. In general, this refers to when a developer is unwilling to give control over modification of the code base to the refactoring tool due to perceived potential problems. This can manifest for a number of reasons. The developer may be unfamiliar with the tool and unwilling to risk experimenting with a tool that could modify the program in unexpected ways. The developer may be unfamiliar with the terms the tool uses, or the information it displays, or the tool may be difficult to learn or use. They may not understand exactly what the tool intends to change about their program. They may not know how the tool will affect the style or readability of the code, or they may be familiar with this and knowingly dislike what it will do to their code. Pinto et al. [19] found that some developers will avoid suggested refactorings if they would need to trade readability for atomicity. In any of these scenarios, a more trustworthy option for the developer would be to rely on their own intuition, abilities, and experience.

Developers also reported concerns that refactoring tools would implement poor design choices, either due to bugs in the tool, inconsistencies with the detection algorithms, or special cases with the code base, such as reflection. Several popular refactoring tools have been shown to contain such bugs that modify program behavior without the developer ever knowing [26, 1]. Veracity, or the extent to which refactorings can be trusted, is emerging problem from big data perspective.

4.4 Behaviour Preservation

Today, a wide variety of refactoring tools automates several aspects of refactoring. However, ensuring the behavior preserving property when building toolassisted refactoring is challenging. Several formalisms and techniques have been proposed in the existing literature to guarantee the behavior preservation and correctness of refactorings. Actual source code transformation and a set of preconditions are the two main parts for any refactoring operation to be performed by automated refactorings.

5 Refactoring Recommendation

Performing refactoring in a large software system can be very challenging. While refactoring is being applied by various developers [5], it would be interesting to

evaluate their refactoring practices. We would like to capture and better understand the code refactoring best practices and learn from these developers so that we can recommend them for other developers. There is a need to build a refactoring recommendation system to (1) identify refactoring opportunities and pinpoint design flaws, and (2) apply refactoring solutions. To support future refactorings, structural, semantic, dynamic, and historical information between code components need to be considered. Recent proposed recommenders do generate a large list of refactorings to apply. This represents a challenge for practitioners, since they do not want to lose the identity of their design, also they cannot fully understand the impact of such large set of code changes. Such a problem is mapped to big data Volume and Veracity. Furthermore, running such set of refactoring, requires handling several constraints. it is to satisfy the correctness of the applied refactorings. Previous studies distinguish between two kinds of constraints: structural constraints and semantic constraints. Structural constraints were extensively investigated in the literature. Fowler, for example, defined in [10] a set of pre and post-conditions for a large list of refactoring operations to ensure the structural consistency. However, software engineers should check manually all actors related to the refactoring operation to inspect the semantic relationship between them. In the next subsections, we further detail the challenges of establishing the relationship between refactoring and its corresponding target code element(s).

5.1 Structural Relationship

Structural relationships mean selecting quality metrics to measure system improvement before and after the application of refactoring that includes method calls, shared instance variables, or inheritance relationships. Several quality metrics have been reported in the literature to capture different aspects of internal quality attributes. For example, the coupling between object (CBO) metric correlates with coupling, i.e., the higher the CBO value, the higher the coupling between classes.

5.2 Semantic Relationship

To determine the semantic relationship between code components, textual similarity is measured. If the terms of two code components (i.e., class or method) are very similar, then it is probable that developers used the same terms to express the responsibilities implemented by the class or the method. For example, two methods are considered conceptually related if both of these methods perform conceptually similar actions. This information is useful for grouping similar code components together. There are a few quality metrics to capture semantic similarity (e.g., the conceptual cohesion of classes (C3) and the conceptual coupling between classes (CCBC)). For example, in order to recommend Move Class refactoring, software module classes having high CCBC values can be grouped together. Consequently, the changes can be localized easily by developers and the software will be more manageable and maintainable.

5.3 Historical Information

The refactoring process can be automated, not only by using the state-of-the-art features (improving design metrics and quality attributes) but also with contextual features that simulate developers' presence by using refactoring operations previously performed by developers. These refactoring operations could be obtained by using refactoring-mining tools such as RefactoringMiner and RefDiff that identify refactoring applied between two subsequent versions of a software system.

6 Refactoring Visualization

Visualizing refactoring activity applied to the source code helps provide a big picture about refactoring. It helps gain insight about the source code and improves the understandability of the software. However, visualizing large refactoring activity presents both technical and cognitive challenges. Particularly, if the code change is complex and large, the task of detecting refactoring anomalies and looking for defects becomes more challenging. Developers could perform batch refactoring or sequence of refactoring operations. Murphy-Hill et al. [17] define batch refactorings as refactoring operations that are executed within 60 seconds of each other. Their findings show that developers repeat the application of refactoring, and 40% of refactorings performed using a refactoring tool occur in batches. Recently, Brito et al. [7] introduced a refactoring graph concept to assess refactoring over time. The authors analyzed 10 Java projects, extracted 1,150 refactoring subgraphs, and evaluated their properties: size, commits, age, composition, and developers. To increase the trust between developers and the tool, Bogart et al. [6] recently extended JDeodorant tool by providing developers with the possibility of verifying their refactoring outcomes. The extended tools provide timely visualization of multiple selected refactorings, and detects whether there is a conflict or not.

Visualizing big data refactoring is not deeply studied or discussed in the refactoring literature. Refactoring visualization is a vital process since it allows developers to look at the code and learn how it is organized and how it works. Further, it assists developers in pinpointing possible bad code smells that violate design principles, determining which code paths are susceptible to a bug, and saving development time.

Research in refactoring should expand on refactoring graphs at the method level, and focus on class and package level refactorings. Also, research could complement existing git-based (e.g., RefactoringMiner [24] and RefDiff [21]) and contemporary IDE refactoring tools (e.g., JDeodorant [23] and RefFinder [20]) with visualization features.

7 Conclusion

In this chapter, we have explored various challenges that the rise of refactoring research has been facing, and which represent interesting research opportunities

12 Eman Abdullah AlOmar, Mohamed Wiem Mkaouer, and Ali Ouni

for the big data community. For each refactoring challenge, we explored its related studies to understand its growth and complexity, then we discussed how it is linked to big data dimensions. As we established stronger connections between refactoring and big data, we hope to see emerging studies leveraging big data techniques and frameworks to take refactoring research to the next level.

Bibliography

- Rafi Almhana, Wiem Mkaouer, Marouane Kessentini, and Ali Ouni. Recommending relevant classes for bug reports using multi-objective search. In 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 286–295. IEEE, 2016.
- [2] Eman Abdullah Alomar, Mohamed Wiem Mkaouer, and Ali Ouni. Can refactoring be self-affirmed? an exploratory study on how developers document their refactoring activities in commit messages. In *Proceedings of* the 3rd International Workshop on Refactoring, New York, NY, USA, 2019. ACM.
- [3] Eman Abdullah AlOmar, Mohamed Wiem Mkaouer, and Ali Ouni. Toward the automatic classification of self-affirmed refactoring. *Journal of Systems* and Software, page 110821, 2020.
- [4] Eman Abdullah AlOmar, Mohamed Wiem Mkaouer, Ali Ouni, and Maroune Kessentini. On the impact of refactoring on the relationship between quality attributes and design metrics. In 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pages 1–11. IEEE, 2019.
- [5] Eman Abdullah AlOmar, Anthony Peruma, Christian D Newman, Mohamed Wiem Mkaouer, and Ali Ouni. On the relationship between developer experience and refactoring: An exploratory study and preliminary results. In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, pages 342–349, 2020.
- [6] Alex Bogart, Eman Abdullah AlOmar, Mohamed Wiem Mkaouer, and Ali Ouni. Increasing the trust in refactoring through visualization. In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, pages 334–341, 2020.
- [7] Aline Brito, Andre Hora, and Marco Tulio Valente. Refactoring graphs: Assessing refactoring over time. arXiv preprint arXiv:2003.04666, 2020.
- [8] Dustin Campbell and Mark Miller. Designing refactoring tools for developers. In Proceedings of the 2nd Workshop on Refactoring Tools, page 9. ACM, 2008.
- [9] Danny Dig, Can Comertoglu, Darko Marinov, and Ralph Johnson. Automated Detection of Refactorings in Evolving Components, pages 404–428.
 Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
- [10] Martin Fowler, Kent Beck, John Brant, William Opdyke, and don Roberts. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
- [11] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722, 2014.

- 14 Eman Abdullah AlOmar, Mohamed Wiem Mkaouer, and Ali Ouni
- [12] Shinpei Hayashi, Yasuyuki Tsuda, and Motoshi Saeki. Search-based refactoring detection from source code revisions. *IEICE TRANSACTIONS on Information and Systems*, 93(4):754–762, 2010.
- [13] Miryung Kim, Thomas Zimmermann, and Nachiappan Nagappan. A field study of refactoring challenges and benefits. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, page 50. ACM, 2012.
- [14] Mika V Mäntylä and Casper Lassenius. Drivers for software refactoring decisions. In Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, pages 297–306. ACM, 2006.
- [15] Emerson Murphy-Hill. Programmer friendly refactoring tools. 2009.
- [16] Emerson Murphy-Hill and Andrew P Black. Refactoring tools: Fitness for purpose. *IEEE software*, 25(5), 2008.
- [17] Emerson Murphy-Hill, Chris Parnin, and Andrew P Black. How we refactor, and how we know it. *IEEE Transactions on Software Engineering*, 38(1):5– 18, 2011.
- [18] Stas Negara, Nicholas Chen, Mohsen Vakilian, Ralph E Johnson, and Danny Dig. A comparative study of manual and automated refactorings. In *European Conference on Object-Oriented Programming*, pages 552–576. Springer, 2013.
- [19] Gustavo H Pinto and Fernando Kamei. What programmers say about refactoring tools?: An empirical investigation of stack overflow. In *Proceedings* of the 2013 ACM workshop on Workshop on refactoring tools, pages 33–36. ACM, 2013.
- [20] K. Prete, N. Rachatasumrit, N. Sudan, and M. Kim. Template-based reconstruction of complex refactorings. In 2010 IEEE International Conference on Software Maintenance, pages 1–10, Sept 2010.
- [21] D. Silva and M. T. Valente. Refdiff: Detecting refactorings in version histories. In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pages 269–279, May 2017.
- [22] Liang Tan and Christoph Bockisch. A survey of refactoring detection tools. In Software Engineering, 2019.
- [23] Nikolaos Tsantalis, Theodoros Chaikalis, and Alexander Chatzigeorgiou. Jdeodorant: Identification and removal of type-checking bad smells. In 2008 12th European Conference on Software Maintenance and Reengineering, pages 329–331. IEEE, 2008.
- [24] Nikolaos Tsantalis, Matin Mansouri, Laleh M Eshkevari, Davood Mazinanian, and Danny Dig. Accurate and efficient refactoring detection in commit history. In *Proceedings of the 40th International Conference on Software Engineering*, pages 483–494. ACM, 2018.
- [25] Mohsen Vakilian, Nicholas Chen, Stas Negara, Balaji Ambresh Rajkumar, Brian P Bailey, and Ralph E Johnson. Use, disuse, and misuse of automated refactorings. In *Proceedings of the 34th International Conference on Software Engineering*, pages 233–243. IEEE Press, 2012.
- [26] Mathieu Verbaere, Ran Ettinger, and Oege De Moor. Jungl: a scripting language for refactoring. In Proceedings of the 28th international conference on Software engineering, pages 172–181. ACM, 2006.

[27] Zhenchang Xing and Eleni Stroulia. Umldiff: An algorithm for objectoriented design differencing. In Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, ASE '05, pages 54–65, New York, NY, USA, 2005. ACM.