IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 23, 2021, accepted December 30, 2021, date of publication December 31, 2021,
date of current version June 1, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3140036

What Refactoring Topics Do Developers Discuss? A Large Scale Empirical Study Using Stack Overflow

CHAIMA ABID[1], KHOULOUD GAALOUL[1], MAROUANE KESSENTINI[2], AND VAHID ALIZADEH[3], (Graduate Student Member, IEEE)

[1]Computer and Information Science Department, University of Michigan–Dearborn, Dearborn, MI 48128, USA
[2]School of Engineering and Computer Science, Oakland University, Rochester, MI 48309, USA
[3]College of Computing and Digital Media, DePaul University, Chicago, IL 60614, USA

Corresponding author: Marouane Kessentini (marouane@umich.edu)

ABSTRACT Due to the growing complexity of software systems, there has been a dramatic increase in research and industry demand on refactoring. Refactoring research nowadays addresses challenges beyond code transformation to include, but not limited to, scheduling the opportune time to carry refactoring, recommending specific refactoring activities, detecting refactoring opportunities and testing the correctness of applied refactorings. Very few studies focused on the challenges that practitioners face when refactoring software systems and what should be the current refactoring research focus from the developers' perspective. Without such knowledge, tool builders invest in the wrong direction, and researchers miss many opportunities for improving the practice of refactoring. In this paper, we collected data from the popular online Q&A site, Stack Overflow, and analyzed posts to identify what do developers ask about refactoring. We clustered these questions to find the different refactoring related topics using one of the most popular topic modeling algorithms, Latent Dirichlet Allocation (LDA). We found that developers are asking about design patterns, design and user interface refactoring, web services, parallel programming, and mobile apps. We also identified what popular refactoring challenges are the most difficult and the current important topics and questions related to refactoring. Moreover, we discovered gaps between existing research on refactoring and the challenges developers face. To the best of our knowledge, this paper represents the first Stack Overflow study to identify the refactoring topics discussed by developers. Our study can help researchers to focus on practical refactoring problems, practitioners know more about current challenges and build better refactoring tools, and educators revise curriculum to target current needs on refactoring.

INDEX TERMS Empirical study, stack overflow, refactoring.

I. INTRODUCTION

Refactoring [28], [31], [65] is a technique that improves the design structure while preserving the overall functionality and behavior. It is a key practice in agile development processes and well supported by tools integrated with major IDEs. Refactoring is an extremely important solution to address the challenge of managing software complexity [16], [27], [92], and has experienced tremendous adoption in Object-Oriented systems [14], [17], [23], [29], [30], [36], [38], [45], [48], [62], [68], [87].

The associate editor coordinating the review of this manuscript and approving it for publication was Mahmoud Elish.

A recent study by the US Air Force Software Technology Support Centre (STSC) shows that the code restructuring of several software systems reduced developers' time by over 60% when introducing new features into a restructured architecture [5]. Due to the growing complexity of software systems, the last ten years have seen a dramatic increase and industry demand for tools and techniques on software refactoring.

Drawing from the emerging need for refactoring in new domains (e.g., mobile apps, cloud computing, service architecture, model-driven engineering, etc.), a large number of tools have been proposed covering the refactoring life cycle including the detection of refactoring opportunities [26], [33], [34], [59], [86], when to apply refactorings [49], [53],

recommendation of refactorings [8], [9], [13], [84] and regression testing for refactoring [74], [81]. Several surveys summarize the research progress in the field [3]–[5], [60], [80]. However, identifying the challenges that practitioners face when refactoring received little attention. The few studies on how developers refactor code are limited to surveys with developers at organizations such as Microsoft [46], [47].

Given the current growth of refactoring research with more than 3000 peer-reviewed papers published in the last decade, the gap is growing larger between research and practice. Is the research community paying attention to the needs of developers? What informs the design of new refactoring technology? We believe it is crucially important to understand the current trends in the field, the challenges that developers face when refactoring in the wild, and the most discussed refactoring topics on developer forums. Without such understanding, tool builders invest in the wrong direction, and researchers miss many opportunities for improving the practice of refactoring. We need to understand the new drivers for refactoring innovation from the practitioners' vantage point. In this paper, we performed the first large scale refactoring study on the most popular online Q&A forum for developers, Stack Overflow. We identify the refactoring topics that have been extensively discussed over Stack Overflow and we also draw conclusions and discussions around the most popular topics. Stack Overflow is the most renowned sites dedicated to answering coding questions online. Developers use the forum to seek help and advice from their peers about the technical challenges they face in different development topics. Stack Overflow moderates millions of posts from developers, with different backgrounds, asking questions about a wide range of topics including refactoring. It has a nice and intuitive interface which provides the user with the ability to vote questions up and down. The analysis of the discussed topics in this repository could provide various key insights about the topics of interest to the developers related to refactoring such as the most addressed quality issues, the domains where refactoring is extensively discussed, the preferred level abstractions, the widely addressed anti-patterns, and patterns. Recent studies analyzed Stack Overflow posts in several areas including software security [94], mobile apps [15], [52], [77], and more general programming topics [11], [71] and came up with useful recommendations. We believe applying a similar approach for studying refactoring needs could be equally useful.

The analysis of Stack Overflow refactoring posts is beneficial to developers, researchers, and educators in different ways. Developers can educate themselves about the common issues that others have faced so they can learn about the peer-best-practices. Researchers can use this analysis to understand the real problems faced by programmers in refactoring. Finally, educators may use the result of these analyses to update their courses and focus on the main weaknesses in the background of programmers that may need to be addressed. The mapping between Stack Overflow discussions and existing research topics helps us and others identify the gap between the practitioners and research communities.

Stack Overflow contains more than 42 million posts and associated attributes such as questions, answers, tags that are most representative of the post etc [1], [2]. We first selected the posts related to refactoring by choosing a list of tags such as ''refactoring,'' ''anti-patterns,'' etc. Then, we used an advanced topic model based on, Latent Drichlet Allocation (LDA), to identify the topics. Using this data, we answer the following five research questions:

- **RQ1. What questions and issues related to refactoring are developers discussing?** We found that developers are interested in six main topics related to refactoring which are Creational pattern, Parallel programming, Models Refactor, mobile/UI, SOA, and Design pattern (Section IV-A).
- **RQ2. What are the most popular topics among the questions related to refactoring?** Our results show that Creational Pattern topic has the largest popularity while parallel programming, and mobile/user-interface topics have the lowest (Section IV-B).
- **RQ3. Which refactoring-related topics are the most difficult to answer?** Design patterns topic has the lowest rate of questions with unsatisfactory answers. It also has the lowest average number of views without a relevant answer. The model refactoring is the topic that was the least answered by developers(Section IV-C)
- **RQ4. How do the interests of developers on refactoring topics change over time?** SOA and Design patterns are the refactoring topics that have the highest evolution in the number of questions throughout the years (Section IV-D)
- **RQ5. What are the implications of our empirical study on practitioners, educators, and researchers?** Our study helps researchers focus on practical refactoring problems, practitioners know more about current challenges and build better refactoring tools, and educators revise curriculum to target current needs on refactoring (Section V).

The remainder of this paper is organized as follows. Section 2, dedicated to the current state of refactoring research, sets the context to understand the motivation of our work in Section 3 better. The data collection step and our experimental approach are described in Section 4. Our experimental results are presented and discussed in Sections 5 and 6 respectively. Section 7 concludes the paper.

II. STACK OVERFLOW DATA DESCRIPTION

Stack Overflow is a question and answer website used by beginners as well as professionals belonging to stack-exchange network. It has the largest community compared to the other Q&A websites in the Network. Stack Overflow was launched on September 15, 2008 and it kept growing in popularity. Nowadays more than 17 million questions were

posted in Stack Overflow, and an average of 5956 questions was asked per day in the last four years.

There are currently 54637 tags in Stack Overflow. Among them, the tag ''Javascript'' holds the biggest number of related questions which exceed 1700000 whereas the ''refactoring'' tag helps to identify 6445 questions. Figure 1 shows one example of a refactoring question on Stack Overflow with the title ''Is there a working C++ refactoring tool?.'' Two tags are used for this post: ''refactoring'' and ''C++.'' Furthermore, several metadata are related to a post such as the edit date, the number of views, etc.

To easily access Stack Overflow data, one of the best ways is based on Stack-exchange data-dumps. They represent collections of data archived by the Stack-exchange community. The data is collected yearly and uploaded on the archive.org website. The data-set is divided into several XML files which are Posts.xml, Users.xml, Votes.xml, Comments.xml, PostHistory.xml, and PostLinks.xml. In this study, we have used Posts.xml which contains around 42 million posts. There are five types of posts: Question, Answer, Orphaned Tag Wiki, Tag Wiki Excerpt, and Tag Wiki.

Each type of post can be filtered using the PostTypeID attribute. Besides the PostTypeID, each post has 21 defined attributes which could have value or not depending on the type of the post. Table 1 shows the different attributes defined in the posts.xml file. Of course, one of the main attributes is the tag used to classify the question. We will give details in the next section how we identified the tags related to refactorings to filter the Stack Overflow posts.

III. RESEARCH METHOD

The main goal of this study is to identify the main refactoring-related topics discussed by developers, highlighting the most popular ones and the most difficult ones and understanding developers' interest trends.

We will describe, in this section, the details of the steps adopted to achieve the main goals of this study. In the remainder of the paper, we will use ''document'' to refer to a question and ''corpus'' to refer to the set of questions.

Figure 2 summarizes the different main steps of our Stack Overflow analysis. The first step consists of identifying the list of tags related to refactoring. The second step filters the list of questions using the selected tags. The third main step runs LDA to identify the list of topics related to refactoring by mining the selected questions and answers. Finally, we answered several questions about these topics including trends, difficulty, and evolution over time.

To select the refactoring related documents, we extracted refactoring related tags and filtered the documents dataset using these tags. Then, we pre-processed each document for the LDA topic modeling approach by cleaning it and translating it into a vector of features using the Bag of Words (BOW) representation [54]. We used LDA topic modeling approach because it was widely used in similar problems [11], [77], [94] and it was proven to be able to generate topics that are highly interpretable and provide deep insights to the data.

To filter the questions, we used multiple steps. First, we manually defined an initial list of tags including 10 words: refactoring, design patterns, architecture, anti-patterns, code-cleanup, software-design, software-quality, code-metrics, automated-refactoring. The manual definition of tags is limited and may not cover all the relevant refactoring questions. For instance, some posts are related to refactoring but are not tagged with the refactoring tag in several cases. In order to extract more tags using the initial set of tags, we extracted all the tags defined in Stack Overflow, and for each extracted tag, we assessed to what extent it is relevant and related to the initial tag list. Therefore, we used two heuristics taking inspiration from a similar study [94]. These heuristics are based on $a(t)$: the number of questions that contain both tag t and a refactoring related tag (one of the above 10 words), $b(t)$: number of questions that contains the tag t, and $c(t)$: the number of questions that contain a refactoring related tag (one of the above 10 words).

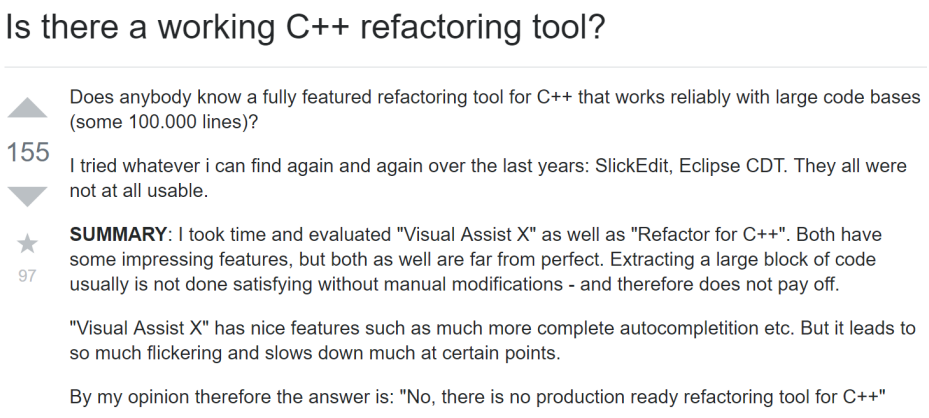
- The first heuristic H1 is defined by the ratio of the number of questions that contain both the tag and a refactoring related tag to the number of questions that contain the tag t. $H1(t) = a(t)/b(t)$
- The second heuristic H2 is the ratio of the number of questions that contain both the tag and a refactoring related tag to the questions identified by the initial set of 10 tags. $H2(t) = a(t)/c(t)$.

We defined thresholds empirically for both heuristics to select relevant tags by trial and error: 0.08 for the first heuristic and 0.0004 for the second heuristic. We thus extracted 94 tags shown in the Table 2.

After inspecting these 94 extracted tags, we manually removed 5 tags from the 94 initial list of tags extracted from Stack Overflow which are: OOP, domain-driven design, dao, mvp and lsp. For instance, OOP(object-oriented programming) was one of the tags that we removed as it had the largest number of questions which is 46618, most of them not being related to refactoring. We finally considered 89 tags which we used to extract 105,463 questions for this empirical study. We checked the relevance of these tags being chosen via validating random samples from the included documents to make sure they are all relevant. The full list of considered tags are described in Table 2.

In order to identify discussed refactoring related topics, we have used a topic modeling technique: Latent Dirichlet Allocation (LDA) [6]. Topic modeling is an approach aiming at finding patterns of words in document collections using hierarchical probabilistic models. Topic modeling may be used to classify the documents of the corpus by discovered latent topics. It specifies a procedure by which documents can be generated by choosing a distribution over topics. Each topic is a distribution that defines how likely each word may appear in a given topic. For more details about LDA, the reader can refer to [6].

As the LDA model is expecting a frequency-weighted document-term matrix, we performed the following steps:



Is there a working C++ refactoring tool?

155

97

Does anybody know a fully featured refactoring tool for C++ that works reliably with large code bases (some 100.000 lines)?

I tried whatever i can find again and again over the last years: SlickEdit, Eclipse CDT. They all were not at all usable.

SUMMARY: I took time and evaluated "Visual Assist X" as well as "Refactor for C++". Both have some impressing features, but both as well are far from perfect. Extracting a large block of code usually is not done satisfying without manual modifications - and therefore does not pay off.

"Visual Assist X" has nice features such as much more complete autocompletition etc. But it leads to so much flickering and slows down much at certain points.

By my opinion therefore the answer is: "No, there is no production ready refactoring tool for C++"

FIGURE 1. Refactoring post found on Stack Overflow.

TABLE 1. Attributes describing a Stack Overflow post.

Attribute	Description
Id	represent a unique id for posts
PostTypeId	Digit that define the type of the post
AcceptedAnswerId	If the post is a question and there is accepted Answer to this question this field will contain the accepted Answer id
ParentId	If the post is an answer this field contains the question id of that answer
CreationDate	This field contains the date time creation of the post eg.:"2008-09-06T08:07:10.730"
DeletionDate	it is defined if the post was deleted and it has the same form as the creation date
Score	this is an integer that represents the upvotes giving to the post. It represents how helpful was the post
ViewCount	This is defined for questions, and it represents how many people viewed the post
Body	Text of the posts
OwnerUserId	The id of the post owner
OwnerDisplayName	the name of the post owner
LastEditorUserId	Id of the last user that modified the post
LastEditorDisplayName	the name of the user that modified the post
LastEditDate	The date of the last post update
LastActivityDate	The date of last Activity related to this post
Title	The title of the post is not an answer
Tags	comma separated strings that list all the tags for the post (defined only for a question)
AnswerCount	defined for a question represent the number of answers related to this question
CommentCount	represent the number of comments for the post it is defined for the question and answer
FavoriteCount	defined only for question post, and it represents the number of users that liked the post
ClosedDate	Defined if moderators of the website closed the post
CommunityOwnedDate	the date when the post was converted to community wiki

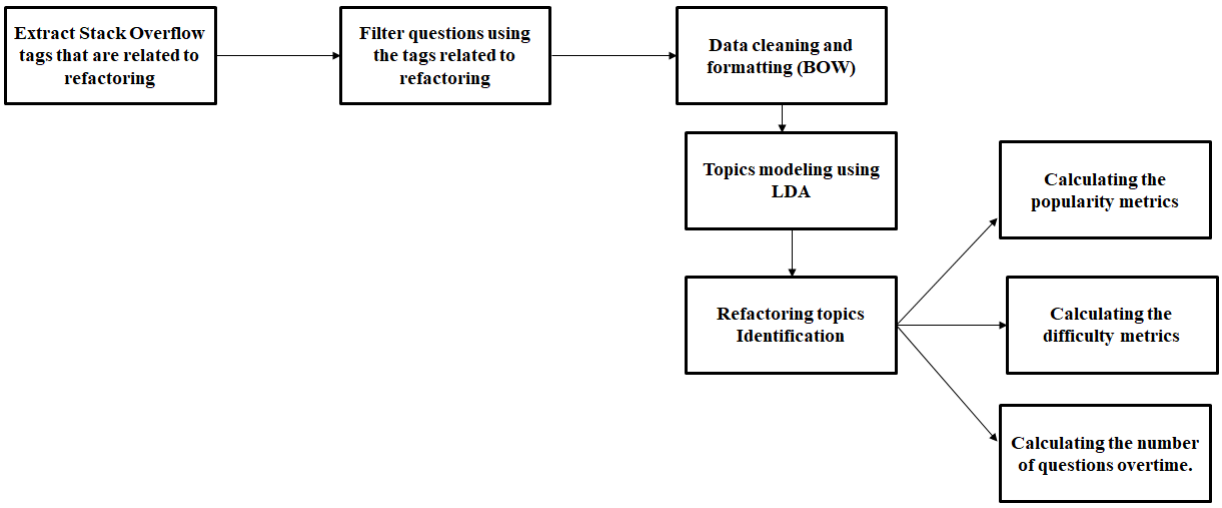


FIGURE 2. An overview of our Stack Overflow analysis.

TABLE 2. List of candidate tags.

design-patterns	design	architecture	singleton	refactoring	microservices
decorator	repository-pattern	factory	scalability	soa	dry
observer-pattern	builder	cqrs	composition	data-access-layer	dto
software-design	factory-pattern	unit-of-work	class-design	abstraction	composite
solid-principles	idioms	modularity	n-tier	business-logic	strategy-pattern
code-duplication	separation-of-concerns	object-oriented-analysis	n-tier-architecture	code-cleanup	legacy-code
methodology	visitor	anti-patterns	ddd-repositories	service-layer	service-locator
3-tier	srp	bridge	domain-model	facade	decoupling
conceptual	automated-refactoring	command-pattern	mediator	maintainability	code-readability
design-principles	visitor-pattern	code-metrics	project-planning	ooad	code-smell
cyclomatic-complexity	module-pattern	application-design	onion-architecture	abstract-factory	business-logic-layer
loose-coupling	factory-method	cocoa-design-patterns	coupling	software-quality	builder-pattern
revealing-module-pattern	clean-architecture	system-design	code-design	law-of-demeter	data-transfer-objects
open-closed-principle	chain-of-responsibility	template-method-pattern	multi-tier	proxy-pattern	architectural-patterns
n-layer	flyweight-pattern	memento	prototype-pattern	gang-of-four	

- First, we aggregated the values from the title, the body and the tag attributes and we removed all the useless meta-data.
- Second, we removed the code snippets and all the HTML tags.
- Third, we tokenized and removed any useless special characters like punctuation and characters that do not belong to English alphabet except for '_' and '-' which are used to join two relevant words together.
- Fourth, we removed stop-words: very common words used in the English language which are not relevant for the clustering of the documents; for example ('do,' 'like,' 'what,' 'I,' 'they,' . . .). Thus, we used the stop-word list provided by NLTK [55] and we added other stop-words that are not relevant for the clustering of refactoring related questions. We also removed words containing less than two characters.
- The fifth step was mainly for normalization based on lemmatization of words which reduces the noise in the data by removing inflectional endings and to return the base or dictionary form of a word, which is known as the lemma [10].
- Finally, we used an automated approach to determine the vocabulary words that we will use as features for BOW (Bag-of-words) representation. The technique consists of calculating the portion of documents that contain a specific word. Then, based on two thresholds we eliminated the very rare keyword that appears in less than 1% of the documents and the very frequent ones that appear in more than 80% of the documents as used in another similar study [94]. We translated the corpus into a TF-IDF matrix. The dimension of the matrix is M * N where M is the number of documents (105463), and N is the number of words in the vocabulary (4872).

The values in the matrix are calculated as TF * IDF:

$$Matrix(d, w) = \frac{frequency(w, d)}{number\_of\_words(d)} * IDF(w) \quad (1)$$

where: w is the corresponding word
d is the corresponding document
frequency(w, d) frequency of w in d
number_of_words(d) number of words in d

$$IDF(w) = log(\frac{\#\_of\_documents}{1 + \#\_of\_documents\_that\_contains\_w})$$

This TF-IDF matrix was used in the LDA model to cluster the questions into topics.

IV. RESULTS

In this section, we summarize the results of the five research questions.

A. RQ1. WHAT QUESTIONS AND ISSUES RELATED TO REFACTORING ARE DISCUSSED BY DEVELOPERS ?

The LDA model identified six main topics discussed by developers. A set of keywords identified each of these topics. To better characterize each topic, we labeled it to match the set of keywords identified by LDA. Table 3 shows the six topics and keywords associated with them sorted by the relevance score of the LDA model.

Most of the words identified by LDA for the first topic are related to object creation: "singleton," "factory," "instance," "constructor" and "create." For the second topic, most of the words refer to parallel programming. This topic includes words like message, request, server, thread and observer. The third topic was related to model refactoring with many keywords about UML diagrams, requirements, and design issues. The fourth topic includes android and other

words related to user interface. In fact, it is normal that most of the questions around Android apps are around refactoring the UI since it is the most crucial part in mobile applications. The fifth topic, service-oriented architecture, includes words like service, user layer, database, and architecture. The high reusability of services in SOA architecture makes refactoring very important to simplify the code and makes it easy to understand. It also helps to achieve modularization at the application level. Finally, the design pattern topic was the last topic with mainly common words like design, pattern, code, singleton, etc. The questions that fall in this topic deal with code standard refactoring, specifically the application of general design pattern to achieve high code quality.

We may highlight that the creational pattern is a specific type of design pattern similar to well-known design patterns like observer and decorator patterns. These patterns were extensively discussed in the refactoring posts on Stack Overflow.

Figure 3 shows the number of questions per dominating topic: it includes questions with a higher probability than 0.5 to belong to a topic based on the LDA output. We notice that the largest number of questions about refactoring is dedicated to SOA architecture. The creational patterns also have a high number of questions even if it is only a sub-type of the design patterns. Although the number of questions about parallel programming was initially small, there is a massive growth in the number of questions asked during the last few years about refactoring for parallel programming.

Figure 4 shows the distribution of the number of questions per dominant topic. According to this figure, 79% of the questions have a dominant topic, and more than 25% of them have 0.8 probability of belonging to their dominating topic. When we have a dominant topic for a question, it does not mean that the question cannot belong to another topic with small probability. Some questions without dominating topic are more likely to belong to more than one topic.

B. RQ2. WHAT ARE THE MOST POPULAR TOPICS AMONG THE QUESTIONS RELATED TO REFACTORING?

In order to assess popularity, we used four metrics. After collecting all the questions related to that topic, we computed:

- the average number of views by exploring the ''ViewCount'' attribute.
- the average number of comments using the CommentCount attribute.
- the average number of favorites using the FavoriteCount attribute.
- the ''Score'' attribute which reflects the relevance of a question to Stack Overflow users, to compute the average score of this set of questions.

It is clear from Figure 5 that the creational pattern topic has the largest average number of views which exceed SOA refactoring despite a large number of questions around refactoring web services. This observation may lead to the conclusion that several of the SOA refactoring related questions did not have a considerable number of views meaning not all questions were relevant or important for refactoring of SOA architecture. However, we still observed more than 1500 views for several of these questions. We have also observed in Figure 5 that most the topics received the same average of number comments and favorites which confirms that all of them are important from practitioners' perspective.

Although the number of questions related to models refactoring is not high, but we clearly see that these few questions are very relevant to practitioners. For instance, the average number of views of a question related to models refactoring exceeds 2000 views which is high compared to the total number of views on the large number of SOA refactoring questions. However, this observation can be balanced based on the number of questions asked per topic since the average number of views may decrease when the number of questions per topic are high (e.g. higher probability for redundancy). Besides, it is clear that refactoring related to parallel programming, and mobile/user-interface topics have the lower popularity since they have the smallest average number of views and the smallest average score compared to the others topics.

C. RQ3. WHICH REFACTORING-RELATED TOPICS ARE THE MOST DIFFICULT TO ANSWER?

We included the answers that are related to the selected refactoring questions. These answers can be tagged as an accepted answer or not. Stack Overflow gives the user who asked a question the ability to accept only one of the answers. To estimate the difficulty, we counted the number of users that found an answer useful (based on the score attribute of the answer) similar to other studies on mining Stack Overflow [15], [19], [94].

We have defined three metrics to estimate difficulty. The first metric is the rate of questions that do not have a relevant answer. For the second metric, we computed the average number of views for unanswered questions in the topic. For the third metric, we calculated the average number of days that are needed to get a relevant answer.

All the results are presented in Figure 6. The number of unanswered question is highly correlated with the number of questions. Thus we presented the ratio of unanswered questions by the total number of questions to ensure a fair comparison between the different topics. First, we can see that most of the questions in a topic have a good percentage of relevant answers. The largest percentage of questions that do not have many relevant answers belong to the refactoring of parallel programming. It may be explained by the challenges associated with making programs running on multiple processors, which is not an easy task. This percentage does not exceed 31% of the questions. We can check from the results that design patterns have the smallest ratio of questions without a relevant answer with a ratio of around 18%. The integration of design patterns into existing architectures using refactorings is not an easy task and requires significant design changes.Thus, it could be challenging and

TABLE 3. The 6 refactoring related topics with the 10 most important words in each topic.

Topic	Words
Creational pattern	singleton, instance, method, factory, java, constructor, create, pattern, call, code
Parallel programming	message, server, observer, microservice, request, time, thread, java, client, connection, performance
Models Refactor	decorator, visitor, decorate, factory, co-evolution, re-design, objects, extract, UML
mobile/UI	Android, view, button, image, presenter, design, page, HTML, text, color
SOA	service, availability, model, coupling, micro-service, layer, repository, interface, database, architecture
Design pattern	design, pattern, principles, hierarchy, reusability, extend

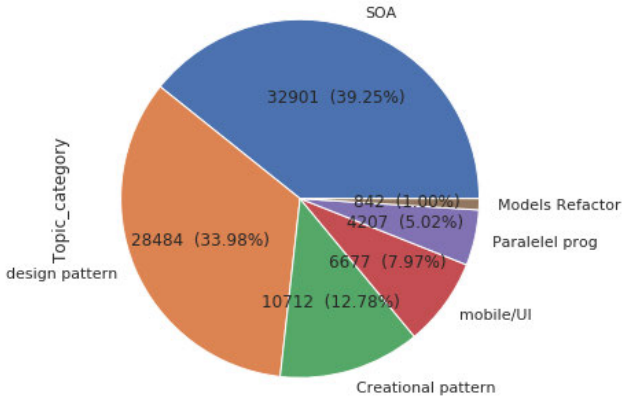


FIGURE 3. Distribution of the number of questions per refactoring topic.

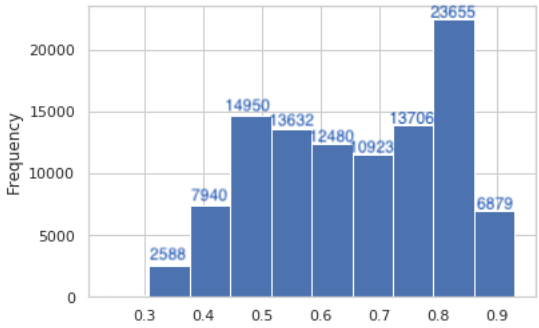


FIGURE 4. The distribution of the number of questions in relation to the probability of the dominant topic.

time-consuming for practitioners to understand and answer these questions.

The same figure shows the average number of views for questions that did not get a relevant answer. This metrics can give us an insight about the difficulty as well. The questions that have no relevant answer got on average more than 200 views for all topics. Thus, it may mean that they are important questions. The ones related to mobile/UI have on average more than 300 views, but no user was able to give a relevant answer which others find it useful. We can conclude that even when a large number of developers accessed to these questions, they find it challenging to answer refactoring questions related to mobile and user interface or it may be an indication that most of the developers viewing these questions are not expert and community of Stack Overflow needs to pay more attention to this kind of topics.

Another important aspect is the average number of days to get a relevant answer to a question. We found that models refactoring have the smallest duration compared to the other topics with only 6 days based on Figure 6. The other values are very similar as we can see that the topic that take the longer time to answer is refactoring of SOA with an average of 10 days to get a relevant answer to the question. In addition, it took between 6 to 10 days to get a relevant answer for the other topics. This means that in average developer does not need to wait very long before getting an answer to their questions. However, 10 days could be a long duration to get an accepted answer for refactoring related questions. Thus, many developers could have moved on and found another solution or abandon the refactoring step because of this long time to get a relevant answer.

Finally, we presented the ratio of the average number of answers to the average number of views. This metrics represents how many answers did the question get compared to the number of views. When this metric is high, it means that many developers can provide an answer to that specific question. It is clear that more than 10% of the people viewing design pattern topic answer that topic. The same observation is valid for SOA architecture. However, developers seem not very interested in answering questions around the model refactoring topic.

D. RQ4. HOW THE INTERESTS OF DEVELOPERS ON REFACTORING TOPICS CHANGE OVER TIME?

For this research question, we investigated the evolution of the number of asked questions throughout the years. We calculated the number of questions of each topic yearly and the evolution is presented in Figure 7. This evolution is related to different refactoring topics. It does not reflect the popularity as a question can be viewed much more times in the future compared to the year where it was asked.

It is clear that throughout the years from 2008 to 2011, refactoring of SOA have seen a significant evolution in the number of questions throughout the years but then there is a very important decrease in the number of asked questions. The same observation goes for the design patterns. One of the reasons that could lead to this evolution is probably because developers no longer need to ask questions since they are already found their questions answered on Stack Overflow. One important observation from this figure is the evolution of refactoring for parallel programming, and the number of questions is still increasing. Before 2016,

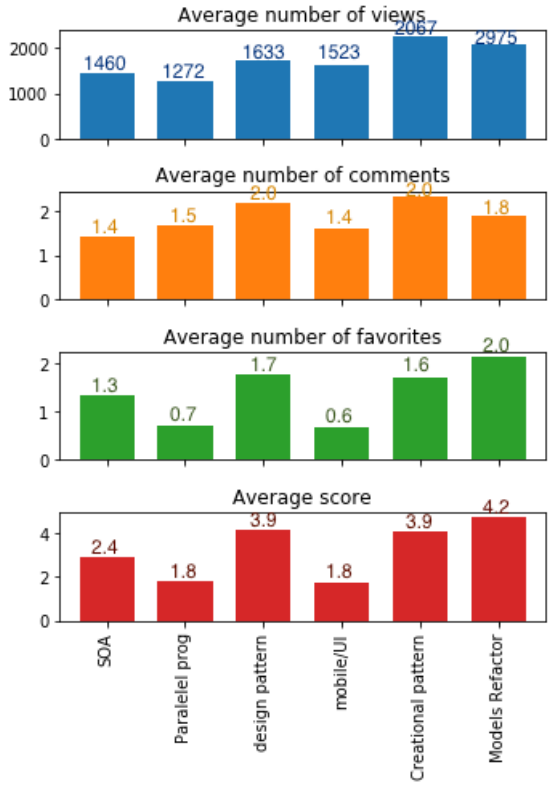


FIGURE 5. The four metrics used to estimate refactoring topics popularity.

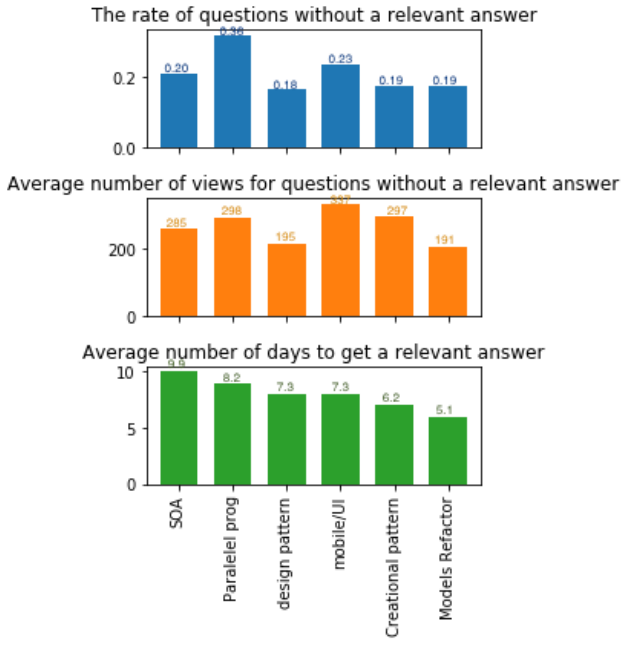


FIGURE 6. The three used metrics to estimate the level of difficulty.

the number of yearly asked question for the refactoring of parallel programming was less than both Mobile/UI and creational pattern. However, we can observe that in 2016 the number of questions asked about refactoring for parallel programming exceeded the refactoring of mobile app and user interface. In 2017, it exceeded the number of questions that are asked about creational design patterns. Thus, developers are showing high interest recently to refactoring for parallel programming.

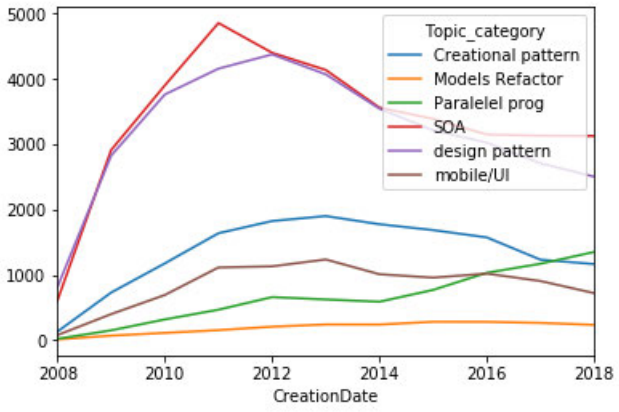


FIGURE 7. The evolution of the number of questions by topic overtime.

V. IMPLICATIONS OF THIS STUDY

We summarize, in this section, the main implications out of our study for researchers, educators and practitioners.

A. IMPLICATIONS FOR RESEARCHERS

Refactoring now expands beyond code-restructuring and targets different artefacts (architecture, model, requirements, etc.) [14], [17], [23], [29], [30], [36], [38], [40], [45], [48], [62], [68], [87], is pervasive in many domains beyond the object-oriented paradigm (cloud computing, mobile, web, etc.) [7], [32], [39], [57], [58], [66], [67], [88]–[90], is widely adopted in industrial settings [25], [37], and the objectives expand beyond improving design into other non-functional requirements (e.g., improve performance, security, etc) [18], [22], [23], [40], [44], [83], [93].

It is clear that the focus of the refactoring research community nowadays goes beyond code transformation to include, but not limited to, scheduling the opportune time to carry refactoring [24], [51], [79], [91], recommending specific refactoring activities [21], [22], [24], [40], [41], [43], [56], [61], [63], [69], [70], inferring refactorings from the code [47], [48], [76], and testing the correctness of applied refactorings [20], [50], [63]. Therefore, the refactoring research efforts are fragmented over several research communities, various domains, and different objectives.

It is clear that there are many intersections between the researchers and practitioner's topics especially in emerging fields such as SOA, Mobile apps, model-driven engineering, and parallel programming. The main surprising outcome is that there are few discussions on Stack Overflow around refactoring for security purposes while it is a growing research topic in academia. Another interesting outcome is related to design patterns. While the academic community is mainly interested in using refactoring to fix anti-patterns, it is clear that practitioners are interested in

integrating patterns using refactoring. The object-oriented paradigm seems to be still a dominant area for refactoring from both researchers and practitioners perspective especially with the increasing interests for model/design refactorings.

This study shows that practitioners are not mainly focusing on JAVA when asking questions on refactoring. However, the current research trends on refactoring are focusing mainly on JAVA. The practitioners are asking more questions on Python while there is a little of tools support and research to refactor Python code. Furthermore, most of existing research studies on refactoring are focusing on the automation of this process while the majority of questions on Stack Overflow are not about automated tools for refactoring but around bugs observed after manually applying refactorings. Thus, the research community may focus more on providing automated regression testing approaches to increase developers trust on applied refactorings.

B. IMPLICATIONS FOR EDUCATORS

Based on the large number of questions asked by practitioners on Stack overflow about refactoring, it is clear that educators need to increase students' awareness and expertise in the evolution of software systems. When students graduate and join the software industry they rarely build software systems from scratch but often spend more time studying and modifying existing systems. Traditionally, students have had a preconceived notion that evolution is a secondary concern. In order to better prepare them for the challenges they will face, we must invest in these curriculum innovations now.

We believe that this study will help educators shift through what's out there and determine the current issues in software quality. They will be able to understand the importance of refactoring and integrating it into education. This makes the students' education in software quality assurance (SQA) in general and in refactoring in particular more efficient and up to date.

While most of SQA courses focus mainly applying refactorings, this study show that practitioners are facing challenges beyond just the execution of refactorings to manage, detect, prioritize and test the refactorings. Thus, educator may think about training the current and next generation of practitioners on the whole refactoring life cycle. Another interesting observation from our study is the large amount of questions about refactoring of Service Oriented Architectures. However, most of existing curricula focuses on JAVA refactoring and Object Oriented design restructuring in general. Thus, educators may consider introducing more background and material related to micro-services migration via refactoring.

C. IMPLICATIONS FOR PRACTITIONERS

Due to the growing complexity of software systems, the last ten years have seen a dramatic increase and industry demand for tools and techniques on software refactoring which is confirmed in our study by the large number of refactoring questions asked by practitioners on Stack overflow.

Our study may help developers be more aware of the importance of writing clean code that follow well defined design patterns. This way, they will be able to prevent the issues that other developers are facing. In addition, they'll be able to know the hot topics in refactoring and therefore what to focus on their self-training efforts. We observed in our study that practitioners are mainly performing refactorings manually. Thus, it is important for them to try some recent semi-automated refactoring tools or prototypes offered for several programming languages rather than spending a lot of energy on the time-consuming and risky manual refactoring.

Another observation is the important focus of developers on introducing design patterns which is an area widely explored in refactoring research. Thus, practitioners may identify some interesting research prototypes that can automated the integration of design patterns. The lack of a refactoring community infrastructure prevents practitioners from using the state-of-the-art advances. They are only aware of refactoring tools that are standard in widely-used IDEs. There is a clear need for an effective communication platform between practitioners and refactoring researchers to identify relevant problems faced by the industry. Practitioners can upload a description of refactoring challenges and provide feedback on existing refactoring tools proposed by researchers.

VI. THREATS TO VALIDITY

Several threats can affect the validity of our results. The first threat is related to the selection of the tags related to refactoring. In fact, we may miss some important tags, but we believe that using the current list of tags we were able to generate an extensive list of questions from Stack Overflow.

The second threat is that not all questions have the appropriate tags since some people could have easily identified a wrong tag to a specific question. Thus, it is possible that we collected some irrelevant questions in our study.

In addition, it is possible that our results may not be generalizable. In this study, we focused on Stack Overflow, which is one of many Q&A websites such as Quora and GitHub. Therefore, our results may not generalize to other Q&A websites. In our future work, we're planning to explore other development communities like GitHub. We will also consider other sources of data in our future work including interviews with practitioners in industry.

In the experiments, we tried many configurations for the LDA model by tuning the probability state of the model and the number of topics. However, these parameters may impact the quality of our results. As for the data cleaning, we can probably introduce more stop_words to reduce the noise in the vocabulary when identifying the refactoring topics.

We believe that the study of the popularity and difficulties of topics is very subjective giving that there is no way to get this measurement directly from the meta-data of the questions. Therefore, we tried to use a combination of metrics to answer these questions, and these metrics could be open to several possible interpretations.

VII. RELATED WORK

A. MINING INFORMATION FROM STACK OVERFLOW POSTS

Stack Overflow was created to help developers with computer programming, but it is becoming a useful knowledge repository for researchers. Therefore, several studies have used Stack Overflow to get an insight into the different questions discussed in practice [78]. Recent studies have focused on mining issues addressed by developers and clustering the related questions [11], [77], [94]. They all used the LDA topic modeling techniques. Yang *et al.* [94] clustered security related questions using a combination of LDA and genetic algorithms. They highlighted the most difficult and most popular security-related questions. Hassan *et al.* [11] adopted LDA to analyze the topics that developers talked about in software engineering, in general, and highlighted the main popular trends in the field such as mobile computing. Rosen *et al.* [77] addressed mobile specific questions and they also used LDA to understand the main challenges faced by mobile developers. Pinto *et al.* [72] performed and empirical investigation of the top-250 most popular questions about concurrent programming on Stack Overflow. They analyzed the text of both questions and answers to extract the dominant topics of discussion using a qualitative methodology. They observed that even though some questions are related to practical problems like fixing bugs etc., most of them are related to understanding basic concepts. Jin *et al.* [35] presented a study of how gamification affects online community members tendencies in terms of response time. They analyzed the distribution gamification-influenced tendencies on Stack Overflow. They defined metrics related to response time to a question post. Results indicate that most members do not undertake in such rapid response activities.

In the software design and refactoring domain, Tian *et al.* [85] conducted a study on developers' conception of Architecture Smells by collecting and analyzing related posts from Stack Overflow. They used 14 terms to extract 207 relevant posts. They used Grounded Theory method to analyze the extracted posts and find out developers' description of Architecture Smells and their causes. They also collected the approaches and tools for detecting and refactoring the different types of Architecture Smells, quality attributes affected by them, and difficulties in detecting and refactoring Architecture Smells. In another preliminary study, Choi *et al.* [19] used Stack Overflow to investigate practitioner's needs for clone detection and analysis and find out whether code clone techniques and tools have met the requirements of programmers. Tahir *et al.* [82] investigated how developers discuss code smells and anti-patterns in Stack Overflow in order to understand their perceptions of these design problems. They applied quantitative and qualitative techniques to analyse posts containing terms related to code smells and anti-patterns. They found out that developers use Stack Overflow to ask for general assessments of code smells or anti-patterns, rather than asking for refactoring solutions. They also noticed that developers usually ask people to check whether their code contain code smells/anti-patterns or not, and therefore, Stack Overflow is often used as crowd-based code smell/anti-pattern detector. Finally, Pinto *et al.* [73] conducted a qualitative and quantitative study to categorize questions from Stack Overflow about refactoring tools. They presented flaws and desirable features in refactoring tools.

Even though all the studies mentioned above tried to mine posts from Stack Overflow to address different problems faced by developers, none of them has looked at the big picture of refactoring to identify the challenges related to refactoring in general faced by practitioners and what could be the current refactoring trends from the developers' perspective.

B. EMPIRICAL STUDIES ON SOFTWARE REFACTORING

Empirical studies on software refactoring investigated the way how developers are refactoring their code. Murphy-Hill *et al.* [64] investigated how developers perform refactorings. Examples of the exploited datasets are usage data from 41 developers using the Eclipse environment and information extracted from versioning systems. Among their several findings, they show that developers often perform floss refactoring, namely they interleave refactoring with other programming activities, confirming that refactoring is rarely performed in isolation.

Peruma *et al.* investigated the potential problems of applying the refactorings in practice and highlight the lack of a clear guidance and a thorough explanation of the refactorings. As a solution, this work identifies the dependencies among different refactoring tasks to improve the refactoring instruments typically used by the tool vendors. Contrary to our approach, the work applies a different mechanism to extract the questions using SOTorrent. Furthermore, the study focuses the analysis on the refactoring-related questions for the accepted and non-accepted answers. It further investigates the varying refactoring needs for different programming languages and different disciplines/topics raised within the Stack Overflow discussion which are different research questions than the ones addressed in this paper. In another study, Sagar, P.S. et Al. propose an approach to build a machine learning model that predicts the refactorings and their types by detecting patterns in metric variations that helps automatically learn the type of refactoring to be applied for a given commit which is a different scope than this paper. Kim *et al.* [47] present a survey of software refactoring with 328 Microsoft engineers to investigate when and how they refactor code and developers' perception towards the benefits, risks, and challenges of refactoring. They show that the major risk factor perceived by developers with regards to refactoring is the introduction of bugs and one of the main benefits they expect is to have fewer bugs in the future, thus indicating the usefulness of refactoring for code components exhibiting high fault-proneness. On top of that, 46% of the developers said that they mostly refactor during bug fixes and new features implementation. Other works on empirical studies for refactoring analyzed the relationship between

refactoring and bugs [12], [42], [75], showing that refactoring activities can be related to introducing bugs.

VIII. CONCLUSION

We performed, in this paper, the first large scale refactoring study on the most popular online Q&A forums for developers, Stack Overflow. We used 89 tags to extract 105463 questions about refactoring. We used the Latent Dirichlet Allocation (LDA) technique to generate the discussed topics in this repository. We found 6 main topics which are ''Creational pattern,'' ''Parallel programming,'' ''Models refactor,'' ''Mobile/UI,'' ''SOA,'' and ''Design pattern.'' The analysis of these topics provided various key insights about the interests of developers related to refactoring such as the most addressed quality issues, the domains where refactoring is extensively discussed, the widely addressed anti-patterns, and patterns. We have also investigated how the interests of developers on refactoring topics change over the years.

In the future, we are planning to expand our data-set to include all the Stack Overflow data available in the ''archive.org'' website. We will also work on a survey with practitioners from multiple programming domains to qualitatively evaluate the outcomes of the Stack Overflow analysis performed in this paper.

REFERENCES

[1] *Stack Exchange Creative Commons Data Now Hosted by the Internet Archive*. Accessed: Apr. 5, 2019. [Online]. Available: https://stackoverflow.blog/2014/01/23/stack-exchange-cc-data-now-hosted-by-the-internet-archive/

[2] *Stack Exchange Data Dump*. Accessed: Apr. 5, 2019. [Online]. Available: https://archive.org/details/stackexchange

[3] M. Abebe and C.-J. Yoo, ''Trends, opportunities and challenges of software refactoring: A systematic literature review,'' *Int. J. Softw. Eng. Appl.*, vol. 8, no. 6, pp. 299–318, 2014.

[4] J. Al Dallal, ''Identifying refactoring opportunities in object-oriented code: A systematic literature review,'' *Inf. Softw. Technol.*, vol. 58, pp. 231–249, Feb. 2015.

[5] J. Al Dallal and A. Abdin, ''Empirical evaluation of the impact of object-oriented code refactoring on quality attributes: A systematic literature review,'' *IEEE Trans. Softw. Eng.*, vol. 44, no. 1, pp. 44–69, Jan. 2018.

[6] R. Alghamdi and K. Alfalqi, ''A survey of topic modeling in text mining,'' *Int. J. Adv. Comput. Sci. Appl.*, vol. 6, no. 1, pp. 132–158, 2015.

[7] A. Ouni, M. Daagi, M. Kessentini, S. Bouktif, and M. M. Gammoudi, ''A machine learning-based approach to detect web service design defects,'' in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 382–391.

[8] V. Alizadeh and M. Kessentini, ''Reducing interactive refactoring effort via clustering-based multi-objective search,'' in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng.*, Sep. 2018, pp. 464–474.

[9] V. Alizadeh, M. Kessentini, W. Mkaouer, M. Ocinneide, A. Ouni, and Y. Cai, ''An interactive and dynamic search-based approach to software refactoring recommendations,'' *IEEE Trans. Softw. Eng.*, vol. 46, no. 1, pp. 932–961, Sep. 2018.

[10] V. Balakrishnan and E. Lloyd-Yemoh, ''Stemming and lemmatization: A comparison of retrieval performances,'' Chalmers Univ., Göteborg, Sweden, Tech. Rep. 38795, 2014.

[11] A. Barua, S. W. Thomas, and A. E. Hassan, ''What are developers talking about? An analysis of topics and trends in stack overflow,'' *Empirical Softw. Eng.*, vol. 19, no. 3, pp. 619–654, 2014.

[12] G. Bavota, B. De Carluccio, A. De Lucia, M. Di Penta, R. Oliveto, and O. Strollo, ''When does a refactoring induce bugs? An empirical study,'' in *Proc. IEEE 12th Int. Workshop Conf. Source Code Anal. Manipulation*, Sep. 2012, pp. 104–113.

[13] G. Bavota, A. D. Lucia, A. Marcus, and R. Oliveto, ''Recommending refactoring operations in large software systems,'' in *Proc. Recommendation Syst. Softw. Eng.* Calgrary, Canada: Springer, 2014, pp. 387–419.

[14] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, ''Clone detection using abstract syntax trees,'' in *Proc. Int. Conf. Softw. Maintenance*, Nov. 1998, pp. 368–377.

[15] S. Beyer and M. Pinzger, ''A manual categorization of Android app development issues on stack overflow,'' in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, Sep. 2014, pp. 531–535.

[16] O. Bjuhr, K. Segeljakt, M. Addibpour, F. Heiser, and R. Lagerstrom, ''Software architecture decoupling at Ericsson,'' in *Proc. IEEE Int. Conf. Softw. Archit. Workshops (ICSAW)*, Apr. 2017, pp. 259–262.

[17] V. S. Adve, D. Dig, S. V. Adve, S. Heumann, R. Komuravelli, J. Overbey, P. Simmons, H. Sung, and M. Vakilian, ''A type and effect system for deterministic parallel Java,'' *ACM Sigplan Notices*, vol. 44, pp. 97–116, Jan. 2009.

[18] Y. Cai, R. Kazman, C. Jaspan, and J. Aldrich, ''Introducing tool-supported architecture review into software design education,'' in *Proc. 26th Int. Conf. Softw. Eng. Educ. Training (CSEE&T)*, May 2013, pp. 70–79.

[19] E. Choi, N. Yoshida, R. G. Kula, and K. Inoue, ''What do practitioners ask about code clone? A preliminary investigation of stack overflow,'' in *Proc. IWSC*, 2015, pp. 49–50.

[20] M. Ó. Cinnéide, D. Boyle, and I. H. Moghadam, ''Automated refactoring for testability,'' in *Proc. IEEE 4th Int. Conf. Softw. Test., Verification Validation Workshops*, Mar. 2011, pp. 437–443.

[21] M. D'Ambros, A. Bacchelli, and M. Lanza, ''On the impact of design flaws on software defects,'' in *Proc. 10th Int. Conf. Qual. Softw.*, Jul. 2010, pp. 23–31.

[22] D. Dig, ''A refactoring approach to parallelism,'' *IEEE Softw.*, vol. 28, no. 1, pp. 17–22, Jan. 2011.

[23] D. Dig, C. Comertoglu, D. Marinov, and R. Johnson, ''Automated detection of refactorings in evolving components,'' in *Proc. ECOOP*, vol. 4067, 2006, pp. 404–428.

[24] B. Du Bois, S. Demeyer, and J. Verelst, ''Refactoring–improving coupling and cohesion of existing code,'' in *Proc. 11th Workshop Conf. Reverse Eng.*, 2004, pp. 144–151.

[25] M. Feathers, *Working Effectively With Legacy Code*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[26] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, ''JDeodorant: Identification and application of extract class refactorings,'' in *Proc. 33rd Int. Conf. Softw. Eng.*, May 2011, pp. 1037–1039.

[27] F. A. Fontana, R. Roveda, M. Zanoni, C. Raibulet, and R. Capilla, ''An experience report on detecting and repairing software architecture erosion,'' in *Proc. 13th Workshop IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Apr. 2016, pp. 21–30.

[28] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Reading, MA, USA: Addison-Wesley, 1999.

[29] M. Fowler, *Refactoring: Improving Design Existing Code*. Reading, MA, USA: Addison-Wesley, 1999.

[30] H. Gall, K. Hajek, and M. Jazayeri, ''Detection of logical coupling based on product release history,'' in *Proc. Int. Conf. Softw. Maintenance*, 1998, pp. 190–197.

[31] G. William Griswold, ''Program restructuring as aid to software maintenance,'' Ph.D. dissertation, Seattle, WA, USA, 1992.

[32] H. Wang, M. Kessentini, T. Hassouna, and A. Ouni, ''On the value of quality of service attributes for detecting bad design practices,'' in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 242–251.

[33] Y. Higo, S. Kusumoto, and K. Inoue, ''A metric-based approach to identifying refactoring opportunities for merging code clones in a Java software system,'' *J. Softw. Maintenance Evol., Res. Pract.*, vol. 20, no. 6, pp. 435–461, Nov. 2008.

[34] K. Hotta, Y. Higo, and S. Kusumoto, ''Identifying, tailoring, and suggesting form template method refactoring opportunities with program dependence graph,'' in *Proc. 16th Eur. Conf. Softw. Maintenance Reeng.*, Mar. 2012, pp. 53–62.

[35] Y. Jin, X. Yang, R. G. Kula, E. Choi, K. Inoue, and H. Iida, ''Quick trigger on stack overflow: A study of gamification-influenced member tendencies,'' in *Proc. IEEE/ACM 12th Workshop Conf. Mining Softw. Repositories*, May 2015, pp. 434–437.

[36] T. Kamiya, S. Kusumoto, and K. Inoue, ''CCFinder: A multilinguistic token-based code clone detection system for large scale source code,'' *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 654–670, Jul. 2002.

[37] J. Kerievsky, *Refactoring to Patterns*. Reading, MA, USA: Addison-Wesley, 2004.

[38] M. Kessentini, S. Vaucher, and H. Sahraoui, "Deviance from perfection is a better criterion than closeness to evil when identifying risky code," in *Proc. IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2010, pp. 113–122.

[39] M. Kessentini and H. Wang, "Detecting refactorings among multiple web service releases: A heuristic-based approach," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 263–272.

[40] W. Kessentini, M. Kessentini, H. Sahraoui, S. Bechikh, and A. Ouni, "A cooperative parallel search-based software engineering approach for code-smells detection," *IEEE Trans. Softw. Eng.*, vol. 40, no. 9, pp. 841–861, Sep. 2014.

[41] J. Kim, D. Batory, D. Dig, and M. Azanza, "Improving refactoring speed by 10X," in *Proc. 38th Int. Conf. Softw. Eng.*, May 2016, pp. 1145–1156.

[42] M. Kim, D. Cai, and S. Kim, "An empirical investigation into the role of API-level refactorings during software evolution," in *Proc. 33rd Int. Conf. Softw. Eng.*, May 2011, pp. 151–160.

[43] M. Kim, M. Gee, A. Loh, and N. Rachatasumrit, "Ref-finder: A refactoring reconstruction tool based on logic query templates," in *Proc. 18th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2010, pp. 371–372.

[44] M. Kim and D. Notkin, "Discovering and representing systematic code changes," in *Proc. IEEE 31st Int. Conf. Softw. Eng.*, 2009, pp. 309–319.

[45] M. Kim, V. Sazawal, and D. Notkin, "An empirical study of code clone genealogies," in *Proc. 10th Eur. Softw. Eng. Conf. Held Jointly*, 2005, pp. 187–196.

[46] M. Kim, T. Zimmermann, and N. Nagappan, "A field study of refactoring challenges and benefits," in *Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng.*, 2012, p. 50.

[47] M. Kim, T. Zimmermann, and N. Nagappan, "An empirical study of refactoring challenges and benefits at Microsoft," *IEEE Trans. Softw. Eng.*, vol. 40, no. 7, pp. 633–649, Jan. 2014.

[48] S. Kim and M. D. Ernst, "Prioritizing warning categories by analyzing software history," in *Proc. 4th Int. Workshop Mining Softw. Repositories (MSR: ICSE Workshops)*, May 2007, p. 27.

[49] S. Lee, G. Bae, H. S. Chae, D.-H. Bae, and Y. R. Kwon, "Automated scheduling for clone-based refactoring using a competent GA," *Softw., Pract. Exper.*, vol. 41, no. 5, pp. 521–550, Apr. 2011.

[50] Y. Lin and D. Dig, "CHECK-THEN-ACT misuse of Java concurrent collections," in *Proc. IEEE 6th Int. Conf. Softw. Test., Verification Validation*, Mar. 2013, pp. 164–173.

[51] Y. Lin and D. Dig, "A study and toolkit of CHECK-THEN-ACT idioms of Java concurrent collections," *Softw. Test., Verification Rel.*, vol. 25, no. 4, pp. 397–425, Jun. 2015.

[52] M. Linares-Vasquez, B. Dit, and D. Poshyvanyk, "An exploratory analysis of mobile development issues using stack overflow," in *Proc. 10th Workshop Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 93–96.

[53] H. Liu, G. Li, Z. Ma, and W. Shao, "Scheduling of conflicting refactorings to promote quality improvement," in *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2007, pp. 489–492.

[54] Y. Liu, Z. Liu, T.-S. Chua, and M. Sun, "Topical word embeddings," in *Proc. AAAI Conf. Artif. Intell.*, 2015, pp. 2418–2424.

[55] E. Loper and S. Bird, "NLTK: The natural language toolkit," 2002, *arXiv:cs/0205028*.

[56] R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," in *Proc. 20th IEEE Int. Conf. Softw. Maintenance*, Sep. 2004, pp. 350–359.

[57] M. Kessentini, H. Wang, J. T. Dea, and A. Ouni, "Improving web services design quality using heuristic search and machine learning," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 410–419.

[58] M. Daagi, A. Ouniy, M. Kessentini, M. M. Gammoudi, and S. Bouktif, "Web service interface decomposition using formal concept analysis," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 171–180.

[59] H. Melton and E. Tempero, "Identifying refactoring opportunities by identifying dependency cycles," in *Proc. 29th Australas. Comput. Sci. Conf.*, vol. 48, 2006, pp. 35–41.

[60] M. Misbhauddin and M. Alshayeb, "UML model refactoring: A systematic literature review," *Empirical Softw. Eng.*, vol. 20, no. 1, pp. 206–251, Feb. 2015.

[61] M. W. Mkaouer, M. Kessentini, S. Bechikh, K. Deb, and M. Ó Cinnéide, "Recommendation system for software refactoring using innovization and interactive dynamic optimization," in *Proc. 29th ACM/IEEE Int. Conf. Automated Softw. Eng.*, Sep. 2014, pp. 331–336.

[62] N. Moha, Y. G. Gueheneuc, L. Duchien, and A. F. L. Meur, "DECOR: A method for the specification and detection of code and design smells," *IEEE Trans. Softw. Eng.*, vol. 36, no. 1, pp. 20–36, Jan. 2010.

[63] E. Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," in *Proc. IEEE 31st Int. Conf. Softw. Eng.*, 2009, pp. 287–297.

[64] E. Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," *IEEE Trans. Softw. Eng.*, vol. 38, no. 1, pp. 5–18, 2012.

[65] F. William Opdyke, "Refactoring object-oriented frameworks," Ph.D. dissertation, Champaign, IL, USA, 1992.

[66] A. Ouni, R. Gaikovina Kula, M. Kessentini, and K. Inoue, "Web service antipatterns detection using genetic programming," in *Proc. Annu. Conf. Genetic Evol. Comput.*, Jul. 2015, pp. 1351–1358.

[67] A. Ouni, M. Kessentini, K. Inoue, and M. O. Cinnéide, "Search-based web service antipatterns detection," *IEEE Trans. Services Comput.*, vol. 10, no. 4, pp. 603–617, Jul. 2015.

[68] A. Ouni, M. Kessentini, and H. Sahraoui, "Search-based refactoring using recorded code changes," in *Proc. 17th Eur. Conf. Softw. Maintenance Reengineering*, Mar. 2013, pp. 221–230.

[69] A. Ouni, M. Kessentini, H. Sahraoui, and M. Boukadoum, "Maintainability defects detection and correction: A multi-objective approach," *Automated Softw. Eng.*, vol. 20, no. 1, pp. 47–79, 2012.

[70] A. Ouni, M. Kessentini, H. Sahraoui, K. Inoue, and K. Deb, "Multi-criteria code refactoring using search-based software engineering: An industrial case study," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 3, pp. 1–53, Jun. 2016.

[71] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and the dynamics of API discussions on stack overflow," Georgia Inst. Technol., Atlanta, GA, USA, Tech. Rep. 1482-004, 2012.

[72] G. Pinto, W. Torres, and F. Castor, "A study on the most popular questions about concurrent programming," in *Proc. 6th Workshop Eval. Usability Program. Lang. Tools*, Oct. 2015, pp. 39–46.

[73] G. H. Pinto and F. Kamei, "What programmers say about refactoring tools: An empirical investigation of stack overflow," in *Proc. ACM Workshop Workshop Refactoring Tools*, 2013, pp. 33–36.

[74] N. Rachatasumrit and M. Kim, "An empirical investigation into the impact of refactoring on regression testing," in *Proc. 28th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2012, pp. 357–366.

[75] J. Ratzinger, T. Sigmund, and H. C. Gall, "On the relation of refactorings and software defect prediction," in *Proc. Int. Workshop Mining Softw. Repositories*, 2008, pp. 35–38.

[76] (Jul. 2014). *Educational Resource for C# Parallel Programmers*. [Online]. Available: http://learnparallelism.net

[77] C. Rosen and E. Shihab, "What are mobile developers asking about? A large scale study using stack overflow," *Empirical Softw. Eng.*, vol. 21, pp. 1192–1223, Jun. 2016.

[78] A. K. Saha, R. K. Saha, and K. A. Schneider, "A discriminative model approach for suggesting tags automatically for stack overflow questions," in *Proc. 10th Workshop Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 73–76.

[79] D. Sahin, M. Kessentini, S. Bechikh, and K. Deb, "Code-smell detection as a bilevel problem," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 1, pp. 1–44, Oct. 2014.

[80] S. Singh and S. Kaur, "A systematic literature review: Refactoring for disclosing code smells in object oriented software," *Ain Shams Eng. J.*, vol. 9, no. 4, pp. 2129–2151, Dec. 2018.

[81] G. Soares, R. Gheyi, and T. Massoni, "Automated behavioral testing of refactoring engines," *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 147–162, Feb. 2013.

[82] A. Tahir, A. Yamashita, S. Licorish, J. Dietrich, and S. Counsell, "Can you tell me if it smells: A study on how developers discuss code smells and anti-patterns in stack overflow," in *Proc. 22nd Int. Conf. Eval. Assessment Softw. Eng.*, Jun. 2018, pp. 68–78.

[83] A. Telea and L. Voinea, "Visual software analytics for the build optimization of large-scale software systems," *Comput. Statist.*, vol. 26, no. 4, pp. 635–654, Dec. 2011.

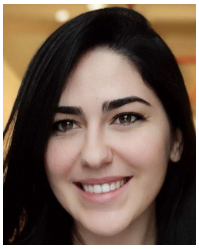
[84] R. Terra, M. T. Valente, K. Czarnecki, and R. S. Bigonha, "Recommending refactorings to reverse software architecture erosion," in *Proc. 16th Eur. Conf. Softw. Maintenance Reeng.*, Mar. 2012, pp. 335–340.

[85] F. Tian, P. Liang, and M. A. Babar, "How developers discuss architecture smells? An exploratory study on stack overflow," in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, Mar. 2019, pp. 91–100.

[86] T. Tourwe and T. Mens, "Identifying refactoring opportunities using logic meta programming," in *Proc. 7th Eur. Conf. onSoftware Maintenance Reeng.*, 2003, pp. 91–100.

[87] N. Tsantalis and A. Chatzigeorgiou, "Identification of move method refactoring opportunities," *IEEE Trans. Softw. Eng.*, vol. 35, no. 3, pp. 347–367, May 2009.
[88] H. Wang, M. Kessentini, and A. Ouni, "Bi-level identification of web service defects," in *Proc. Int. Conf. Service-Oriented Comput.* Paris, France: Springer, 2016, pp. 352–368.
[89] H. Wang, M. Kessentini, and A. Ouni, "Prediction of web services evolution," in *Proc. Int. Conf. Service-Oriented Comput.* Tokyo, Japan: Springer, 2016, pp. 282–297.
[90] H. Wang, A. Ouni, M. Kessentini, B. Maxim, and W. I. Grosky, "Identification of web service refactoring opportunities as a multi-objective problem," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2016, pp. 586–593.
[91] L. Xiao, Y. Cai, and R. Kazman, "Titan: A toolset that connects software architecture with quality analysis," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, Nov. 2014, pp. 763–766.
[92] L. Xiao, "Quantifying architectural debts," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, Aug. 2015, pp. 1030–1033.
[93] A. Yamashita and L. Moonen, "Do developers care about code smells? An exploratory survey," in *Proc. 20th Workshop Conf. Reverse Eng. (WCRE)*, Oct. 2013, pp. 242–251.
[94] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun, "What security questions do developers ask? A large-scale study of stack overflow posts," *J. Comput. Sci. Technol.*, vol. 31, pp. 910–924, Sep. 2016.

CHAIMA ABID is currently pursuing the Ph.D. degree with the Intelligent Software Engineering Group, University of Michigan–Dearborn, under the supervision of Dr. Marouane Kessentini. Her Ph.D. project is concerned with the application of intelligent search and machine learning in different areas, such as web services, refactoring and security. Her current research interests are search-based software engineering, web services, refactoring, security, data analytics, and software quality.



KHOULOUD GAALOUL received the Ph.D. degree from the University of Luxembourg. She was a Postdoctoral Researcher at the SnT Centre for Security, Reliability, and Trust, University of Luxembourg. She is currently a Postdoctoral Researcher with the ISELaboratory, University of Michigan–Dearborn under the supervision of Dr. Marouane Kessentini. Her research interests include model-based software development and analysis of cyber-physical systems, search-based testing, and machine learning. She has been conducting her research in close collaboration with industry partners in the aerospace sector.

MAROUANE KESSENTINI received the Ph.D. degree from the University of Montreal, Canada, in 2012. He is currently a Full Professor at Oakland University. Prior to joining Oakland University, in 2022, he is currently a Tenured Associate Professor. He received several grants from both industry and federal agencies and published over 110 papers in top journals and conferences. He has several collaborations with industry on the use of computational search, machine learning, and evolutionary algorithms to address software engineering and services computing problems. He was a recipient of the Prestigious 2018 President of Tunisia Distinguished Research Award, the University Distinguished Teaching Award, the University Distinguished Digital Education Award, the College of Engineering and Computer Science Distinguished Research Award, four best paper awards, and his AI-based software refactoring invention, licensed and deployed by industrial partners, is selected as one of the top eight inventions at the University of Michigan for 2018 (including the three campuses), among over 500 inventions, by the UM Technology Transfer Office.

VAHID ALIZADEH (Graduate Student Member, IEEE) received the Ph.D. degree from the Intelligent Software Engineering Group, University of Michigan. He is currently a Tenure Track Assistant Professor at DePaul University, Chicago, IL, USA. His Ph.D. project is concerned with the application of intelligent search and machine learning in different software engineering areas, such as refactoring, testing, and documentation. His current research interests include search-based software engineering, refactoring, artificial intelligence, data analytics and software quality.